# Representation of Diffusion Models with applications in Editing, Customization and Recognition Tasks

Rehan Ahmad

Department of Physics, LUMS

## Contents

# 1 Theory of Text-to-Image Diffusion Models

## 1.1 To-Do

1. Thinking in context of embeddings, can we capture the manifold of the embeddings that outputs the most information? You should read the work by Yu ([40]) in more detail.
2. Incorporate Manifold Preserving Guided Diffusion
3. Incorporate Riemannian Diffusion Models

**Capturing Scales of Animals**

I was watching a documentary on Animals and I thought of whether we can devise imagery that naturally capture the scale of these animals. I feel that while many works try to capture spatial relationships, none of them really try to capture this aspect where objects represent the **relative sizes** of what they are.

**Using KV-injection for fine-grained edits**

I was thinking of how KV-injection can effectively perform **fine-grained editing**. In most of the use-cases, it has has been used for appearance transfer but no one has looked into using it for fine-grained edits.

Before we begin laying down the theory of Text-to-Image Diffusion Models, we assume familiarity with the theory of Diffusion Models. See independent research, in particular chapter 9 for an extensive understanding of how these models work. The summary of the loss derivation is given below:

**Summary of the main derivation**

Begin with $\vec{x}_0 \sim q(\vec{x})$ where $q(\vec{x})$ is the dataset of real images. The forward diffusion process progressively adds Gaussian noise to the data $\vec{x}_0$ over T timesteps, producing a sequence of noise data $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T$ :

$$\vec{x}_t \sim q\left(\vec{x}_t \mid \vec{x}_{t-1}\right) = \mathcal{N}\left(\vec{x}_t; \sqrt{1-\beta_t}\vec{x}_{t-1}, \beta_t \mathbb{I}\right) \tag{1.1}$$

Introduce reparameterization technique $\vec{x}_t = \mu + \sigma \odot \epsilon_{t-1}$ to obtain a recursive

process that conditions directly on $\vec{x}_0$ to add noise into the image.

$$\vec{x}_t = \sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon} \tag{1.2}$$

Where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. Therefore, the forward Kernel reads:

$$q\left(\vec{x}_t \mid \vec{x}_0\right) = \mathcal{N}\left(\vec{x}_t; \sqrt{\bar{\alpha}_t}\vec{x}_0, \left(1-\bar{\alpha}_t\right)\mathbb{I}\right) \tag{1.3}$$

We now reverse the diffusion process by aiming to approximate $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$ :

$$\vec{x}_{t-1} \sim p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) = \mathcal{N}\left(\vec{x}_{t-1}; \mu_\theta\left(\vec{x}_t, t\right), \Sigma_\theta\left(\vec{x}_t, t\right)\right) \tag{1.4}$$

Typically, we estimate $\mu_\theta\left(\vec{x}_t, t\right)$ only rather than the set $(\mu_\theta, \Sigma_\theta)$ to simplify the model. $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$ is intractable and so what we aim to is condition on $\vec{x}_0$ such that we approximate:

$$q\left(\vec{x}_{t-1} \mid \vec{x}_0, \vec{x}_t\right) = \mathcal{N}\left(\tilde{\mu}_t\left(\mathbf{x}_t, \boldsymbol{\varepsilon}_t\right), \Sigma_t\left(\mathbf{x}_t, \boldsymbol{\varepsilon}_t\right)\right). \tag{1.5}$$

We obtain:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_t\right) \tag{1.6}$$

$$\tilde{\mu}_t\left(\mathbf{x}_t, \boldsymbol{\varepsilon}_t\right) = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_t\right) \tag{1.7}$$

Subsequently, we aim to minimize the KL divergence between $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$ and $q\left(\vec{x}_{t-1} \mid \vec{x}_0, \vec{x}_t\right)$ :

$$L_t = D_{KL}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$$

for $1 \leq t \leq T-1$. Minimizing the KL divergence of the mean leads to:

$$\begin{aligned}
L_t &= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\varepsilon}}\left[\frac{1}{2\sigma_t^2}\left\|\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \boldsymbol{\varepsilon}_t\right) - \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right)\right\|^2\right] \\
L_t &= \mathbb{E}_{t \sim [1,T], \mathbf{x}_0, \varepsilon_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\mathbf{x}_t, t\right)\right\|^2\right] \\
L_t &= \mathbb{E}_{t \sim [1,T], \mathbf{x}_0, \varepsilon_t}\left[\left\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t, t\right)\right\|^2\right]
\end{aligned} \tag{1.8}$$

We can describe the training and sampling algorithm for DDPM as following:

**Algorithm 1** Training

**repeat**
  $x_0 \sim q(x_0)$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\epsilon \sim \mathcal{N}(0, I)$
  Take gradient descent step on

$$\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2$$

**until** converged

**Algorithm 2** Sampling

$x_T \sim \mathcal{N}(0, I)$
**for** $t = T, \dots, 1$ **do**
  $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$
**end for**
**return** $x_0$

## 1.2 DDIM Sampling

The sampling from Diffusion models as encapsulated by Algorithm 2 can take $T = 1000$ steps where $t = T, \dots, 1$ are the steps with $x_0$ recovering the denoised image. This process leads to long inference times. The hope is to shorten the inference time by generating the denoised image using fewer time steps.

Recall that the steps of Diffusion constitute a Markovian chain. By attempting to sample using fewer time steps, we are essentially looking for a **Non-Markovian** process that can approximate the original diffusion process with reduced computational complexity. Another thing to note is that the sampling methodology as it currently stands is stochastic in nature with $\sigma_t z$ being added at each time-step. A natural question is whether we can eliminate this stochastically to deterministically move towards the manifold in which the data resides.

To motivate such a path in the diffusion space $\mathbb{R}^d$, consider the loss function of (1.8). The structure of the loss function remains unchanged as long as the following relationship holds:

$$\vec{x}_t = \sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t,$$

Because of the this, the loss (1.8) depends upon only on the marginals $q(\vec{x}_t \mid \vec{x}_0)$ rather than the entire chain $q(\vec{x}_{T:1} \mid \vec{x}_0)$ because the term $\sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t$ constructs $\vec{x}_t$ directly from $\vec{x}_0$ and $\epsilon_t$. We thereby look for non-Markovian forward process where instead of considering the entire kernel $q(\vec{x}_{1:T} \mid \vec{x}_0) = \prod_{i=1}^{T} q(\vec{x}_t \mid \vec{x}_{t-1})$, we consider:

$$q_\sigma(\vec{x}_{1:T} \mid \vec{x}_0) = q_\sigma(\vec{x}_T \mid \vec{x}_0) \prod_{t=2}^{T} q_\sigma(\vec{x}_{t-1} \mid \vec{x}_t, \vec{x}_0) \tag{1.9}$$

To maintain the same loss function, we need the same marginal distribution $q(\vec{x}_T \mid \vec{x}_0)$

with the corresponding sampling formula being $\vec{x}_t = \sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t$. Assume that $q_\sigma\left(\vec{x}_{t-1} \mid \vec{x}_t, \vec{x}_0\right)$ is a linear gaussian model, with the mean of $\vec{x}_{t-1}$ depending upon $\vec{x}_t$ and $\vec{x}_0$ linearly and the standard deviation being independent of either of them:

$$q_\sigma\left(\vec{x}_{t-1} \mid \vec{x}_t, \vec{x}_0\right) = \mathcal{N}\left(k_t\vec{x}_t + \lambda_t\vec{x}_0, \sigma_t^2\mathbb{I}\right) \tag{1.10}$$

Parametrizing this, we have:

$$\vec{x}_{t-1} = k_t\vec{x}_t + \lambda_t\vec{x}_0 + \sigma_t^2\bar{\epsilon} \tag{1.11}$$

In the standard Markovian forward pass, we established the following recursion relation:

$$\vec{x}_t = \sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t \tag{1.12}$$

Plugging this inside (1.11):

$$\begin{aligned}
\vec{x}_{t-1} &= k_t\left(\sqrt{\bar{\alpha}_t}\vec{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}\right) + \lambda_t\vec{x}_0 + \sigma_t\bar{\epsilon} \\
\vec{x}_{t-1} &= k_t\sqrt{\bar{\alpha}_t}\vec{x}_0 + \lambda_t\vec{x}_0 + k_t\sqrt{1-\bar{\alpha}_t}\bar{\epsilon} + \sigma_t\bar{\epsilon} \\
\vec{x}_{t-1} &= \left(k_t\sqrt{\bar{\alpha}_t} + \lambda_t\right)\vec{x}_0 + \sqrt{k_t^2\left(1-\bar{\alpha}_t\right) + \sigma_t^2}\bar{\epsilon}
\end{aligned} \tag{1.13}$$

Remember that we need to let the marginal distribution $q\left(\vec{x}_{t-1} \mid \vec{x}_0\right)$ be the same where $q\left(\vec{x}_{t-1} \mid \vec{x}_0\right) = \int q\left(\vec{x}_{t-1} \mid \vec{x}_0\right) q\left(\vec{x}_t \mid \vec{x}_0\right) d\vec{x}_t$. Thus, the recursion relation in equation (1.12) must be satisfied for $\vec{x}_{t-1}$ too. This assumes the following form:

$$\vec{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\vec{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\bar{\epsilon} \tag{1.14}$$

Comparing equation (1.14) and (1.13), we see that the following must be satisfied:

$$\begin{cases} \sqrt{\bar{\alpha}_{t-1}} = k_t\sqrt{\bar{\alpha}_t} + \lambda_t \\ \sqrt{1-\bar{\alpha}_{t-1}} = \sqrt{k_t^2\left(1-\bar{\alpha}_t\right) + \sigma_t^2} \end{cases} \tag{1.15}$$

Solving for $k_t$ and $\lambda_t$ :

$$\begin{aligned}
k_t &= \sqrt{\frac{1-\bar{\alpha}_{t-1}-\sigma_t^2}{\left(1-\bar{\alpha}_t\right)}} \\
\lambda_t &= \sqrt{\bar{\alpha}_{t-1}} - k_t\sqrt{\bar{\alpha}_t} \\
\lambda_t &= \sqrt{\bar{\alpha}_{t-1}} - \sqrt{\frac{1-\bar{\alpha}_{t-1}-\sigma_t^2}{\left(1-\bar{\alpha}_t\right)}}\sqrt{\bar{\alpha}_t}
\end{aligned}$$

$$\lambda_t = \sqrt{\bar{\alpha}_{t-1}} - \sqrt{\frac{\left(1-\bar{\alpha}_{t-1}-\sigma_t^2\right)\bar{\alpha}_t}{\left(1-\bar{\alpha}_t\right)}} \tag{1.16}$$

Therefore, the mean of the distribution in equation (1.10) becomes:

$$\begin{aligned}
\mu &= k_t\vec{x}_t + \lambda_t\vec{x}_0 \\
\mu &= \sqrt{\frac{1-\bar{\alpha}_{t-1}-\sigma_t^2}{\left(1-\bar{\alpha}_t\right)}}\vec{x}_t + \sqrt{\bar{\alpha}_{t-1}}\vec{x}_0 - \sqrt{\frac{\left(1-\bar{\alpha}_{t-1}-\sigma_t^2\right)\bar{\alpha}_t}{\left(1-\bar{\alpha}_t\right)}}\vec{x}_0 \\
\mu &= \sqrt{\bar{\alpha}_{t-1}}\vec{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\vec{x}_t - \sqrt{\bar{\alpha}_t}\vec{x}_0}{\sqrt{1-\bar{\alpha}_t}}
\end{aligned} \tag{1.17}$$

The parametrized family of distribution in equation (1.10) by $\{\sigma_t\}_{1:T}$ becomes the following:

$$q_\sigma\left(\vec{x}_{t-1} \mid \vec{x}_t, \vec{x}_0\right) = \mathcal{N}\left(\sqrt{\bar{\alpha}_{t-1}}\vec{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\vec{x}_t - \sqrt{\bar{\alpha}_t}\vec{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \sigma_t^2\mathbb{I}\right) \tag{1.18}$$

As a result, in the sampling procedure, the updating formula is:

$$\vec{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\vec{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\vec{x}_t - \sqrt{\bar{\alpha}_t}\vec{x}_0}{\sqrt{1-\bar{\alpha}_t}} + \sigma_t\varepsilon \tag{1.19}$$

Rearranging (1.12) for $\vec{x}_0$ and plugging it inside:

$$\begin{aligned}
\vec{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\frac{\vec{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t}{\sqrt{\bar{\alpha}_t}} + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\frac{\vec{x}_t - \sqrt{\bar{\alpha}_t}t\frac{\vec{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t}{\sqrt{\bar{\alpha}_t}}\Big\}}{\sqrt{1-\bar{\alpha}_t}} + \sigma_t\varepsilon \\
\vec{x}_{t-1} &= \sqrt{\bar{\alpha}_{t-1}}\left(\frac{\vec{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_t}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\bar{\epsilon}_t + \sigma_t\varepsilon
\end{aligned} \tag{1.20}$$

Note that the aim of our model is to predict the noise $\bar{\epsilon}_t$:

$$\bar{\epsilon}_t \approx \bar{\epsilon}_\theta^{(t)}$$

Plugging this inside:

$$\vec{x}_{t-1} = \underbrace{\sqrt{\bar{\alpha}_{t-1}}\left(\frac{\vec{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_\theta^{(t)}}{\sqrt{\bar{\alpha}_t}}\right)}_{\text{Predicted } \vec{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma^2}\bar{\epsilon}_\theta^{(t)}}_{\text{Direction pointing to } \vec{x}_t} + \underbrace{\sigma_t\varepsilon}_{\text{Random noise}} . \tag{1.21}$$

Comparing with DDPM, this is generalized form of generative processes. Since

the marginal distribution remains the same, the loss function did not change and the training process is identical. This means that we can use this new generative process with a diffusion model trained in DDPM way and, with different level of $\sigma_t$, we can generate different image with same initial noise. Among different choices, $\sigma_t = 0$ is a special case in which the generation process is deterministic given the initial noise. This model is called denoising diffusion implicit model since it is an implicit probabilistic model and it is trained with the DDPM objective.

For the case $\sigma_t = 0$, the sampling process in equation (1.21) becomes:

$$\vec{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \left( \frac{\vec{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_\theta^{(t)}}{\sqrt{\bar{\alpha}_t}} \right) + \sqrt{1-\bar{\alpha}_{t-1}}\bar{\epsilon}_\theta^{(t)}) \tag{1.22}$$

**DDIM Sampling**

Equation (1.22) represents the seminal result. In terms of the initial state $\vec{x}_0$:, we can write the above as:

$$\vec{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\vec{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\bar{\epsilon}_\theta^{(t)} \tag{1.23}$$

In equation (1.21), if we make the following substitution:

$$\sigma_t^2 = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}\beta_t \tag{1.24}$$

Then DDIM becomes equivalent to the DDPM sampling process. Since we want to obtain $\vec{x}_0$ from $\vec{x}_{t-1}$ in context of denoising, we rearrange equation (1.23):

$$\vec{x}_0 = \frac{\vec{x}_{t-1} - \sqrt{1-\bar{\alpha}_{t-1}}\bar{\epsilon}_\theta^{(t)}}{\sqrt{\bar{\alpha}_{t-1}}})$$

Compared to DDPM, DDIM is able to Generate higher-quality samples using a much fewer number of steps, which was the entire aim to begin with. However, besides that, the models have added benefits, which are:

1. **Consistency Property**: DDIMs introduce a deterministic generative process in which $\sigma z_t = 0$ This deterministic process means that for a given latent variable, the steps to generate a sample are fixed and do not involve any randomness. As a result, the generated samples will be consistent across different runs if conditioned on the same latent variable.

2. **Semantically Meaningful Interpolation**:Due to the deterministic nature of

DDIMs, the consistency in the generated samples allows for smooth and meaningful interpolation between different latent variables.

### DDIM as a Neural ODE

Consider the main result of DDIM:

$$\vec{x}_{t-\Delta t} = \sqrt{\bar{\alpha}_{t-\Delta t}} \left( \frac{\vec{x}_t - \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_\theta^{(t)}}{\sqrt{\bar{\alpha}_t}} \right) + \sqrt{1-\bar{\alpha}_{t-\Delta t}}\bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$

This equation represents a discrete update step for the variable $\vec{x}_{t-1}$. We can actually represent this in continuous paradigm as the authors of the paper note in subsection 4.3. To being doing so, let us distribute $\sqrt{\bar{\alpha}_{t-1}}$ :

$$\vec{x}_{t-\Delta t} = \sqrt{\frac{\bar{\alpha}_{t-\Delta t}}{\bar{\alpha}_t}}\vec{x}_t - \sqrt{\bar{\alpha}_{t-1}}\frac{\sqrt{1-\bar{\alpha}_t}\bar{\epsilon}_\theta^{(t)}}{\sqrt{\bar{\alpha}_t}} + \sqrt{1-\bar{\alpha}_{t-\Delta t}}\bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$

$$\vec{x}_{t-\Delta t} = \sqrt{\frac{\bar{\alpha}_{t-\Delta t}}{\bar{\alpha}_t}}\vec{x}_t + \left( \sqrt{1-\bar{\alpha}_{t-\Delta t}} - \sqrt{\frac{\bar{\alpha}_{t-\Delta t}\left(1-\bar{\alpha}_t\right)}{\bar{\alpha}_t}} \right) \bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$

Divide by $\sqrt{\bar{\alpha}_{t-\Delta t}}$ :

$$\frac{\vec{x}_{t-\Delta t}}{\sqrt{\bar{\alpha}_{t-\Delta t}}} - \sqrt{\frac{1}{\bar{\alpha}_t}}\vec{x}_t = \left( \sqrt{\frac{1-\bar{\alpha}_{t-\Delta t}}{\bar{\alpha}_{t-\Delta t}}} - \sqrt{\frac{1-\bar{\alpha}_t}{\bar{\alpha}_t}} \right) \bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$

Notice how similar to a differential the entire thing looks. The right hand and left hand appear to be change in the following parameter:

$$\begin{cases} \tau = \sqrt{\frac{1-\bar{\alpha}_t}{\bar{\alpha}_t}} \\ \vec{u} = \sqrt{\frac{1}{\bar{\alpha}_t}}\vec{x}_t \end{cases}$$

In that case, we have:

$$\vec{u}(t-\Delta t) - \vec{u}(t) = (\tau(t-\Delta t) - \tau(t))\bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$

The derivative is:

$$\frac{\tau(t-\Delta t) - \tau(t)}{\Delta t} = \frac{d\tau}{dt} \text{ and } \frac{\tau(t-\Delta t) - \tau(t)}{\Delta t} = \frac{du}{dt}$$

Plugging this inside the above equation:

$$\frac{d\vec{u}}{dt}\Delta \mathrm{t} = \frac{d\tau}{dt}\Delta \mathrm{t}\bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$
$$\frac{d\vec{u}}{dt} = \frac{d\tau}{dt}\bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$
$$d\vec{u} = d\tau\bar{\epsilon}_\theta^{(t)}(\vec{x}, t)$$

Finally, note that $\vec{x}_t = \frac{\vec{u}}{\sqrt{1+\tau^2}}$. Thus, we can write in continuous time regime:

$$d\vec{u} = d\tau(t)\bar{\epsilon}_\theta^{(t)}\left(\frac{\vec{u}}{\sqrt{1+\tau^2}}, t\right) \tag{1.25}$$

Equation (1.25) allows to view DDIM sampling as an Euler scheme for solving Equation (1.25) with initial condition $\vec{u}(t = T) \sim \mathcal{N}(0, \alpha_T \mathbb{I})$. This illustrates that we can use fewer sampling steps during inference than the value of T chosen during training by using a coarser discretization of the ODE. On introducing this methodology, the authors note the following thing:
With enough discretization steps, we can also reverse the generation process (going from t = 0 to T , that is from the image to the noisy gaussian), which encodes $\vec{x}_0$ to $\vec{x}_T$ and simulates the reverse of the ODE in Eq. (1.25). This suggests that unlike DDPM, we can use DDIM to obtain encodings of the observations (as the form of $\vec{x}_T$ ), which might be useful for other downstream applications that requires latent representations of a model.

## 1.3 DDIM Inversion

DDIM inversion involves beginning with an observed image $x_0$ and applying the reverse DDIM process to retrieve its latent representation $x_T$. This method can be applied in contexts such as image editing where modifications to specific regions of the image are desired while preserving the overall structure. By transforming the image into its latent representation, it becomes possible to manipulate specific features in the latent space which leads to controlled edits.

The DDIM sampling process is defined by equation (1.23). The underlying dynamics are governed by the equation (1.25). Subsequently, the reverse process follows the corresponding inverse ODE:

$$du = -d\tau(t)\,\hat{\epsilon}_\theta^{(t)}\left(\frac{u}{\sqrt{1+\tau^2}}, t\right). \tag{1.26}$$

This captures the system's behavior as it evolves in reverse. Essentially, DDIM sampling can be understood as an Euler approximation for solving this differential

equation with the initial condition being $u(t = T) \sim \mathcal{N}(0, \alpha_T \mathbf{I})$. By utilizing fewer sampling steps during inference, one can adjust the discretization granularity of the ODE. Through sufficient discretization steps, it becomes feasible to reverse the generation process, transitioning from $t = 0$ to $T$.

**Inversion in Latent Diffusion Models**

Consider a given image $I$. Usually in Img-to-Img framework involving Latent Diffusion models, $I$ is first mapped to a latent representation using VQ-VAE, yielding $z_0 = E(I)$. This latent representation is subsequently noised via a forward diffusion step to produce $z_T \sim \mathcal{N}(0, \mathbf{I})$, without involving U-Net. The reverse diffusion process then denoises $z_T$ back to $z_0$, using U-Net.
By contrast, DDIM inversion begins with mapping $I$ to $z_0 = E(I)$. Rather than using the forward process to introduce noise, the U-NeT is used to carry out the reverse diffusion process as defined by (1.26). Here, the U-Net gradually introduces noise to generate $z_T \sim \mathcal{N}(0, \mathbf{I})$. Unlike the stochastic nature of the Img-to-Img pipeline, the DDIM approach is deterministic which ensures that the latent representations encodes the semantic content of the original image $I$

## 1.4 Guidance in Diffusion models

While training generative models on images with conditioning information, it is common to generate samples conditioned on class labels or a piece of descriptive text. Since we are concerned with the latter task with T2I diffusion models being conditioned on text, we take guidance as the subject of our study in this subsection. Until now, we had been learning just to estimate the underlying probability distribution $f_\phi(\vec{x}_t, t)$ rather an explicit map between the labels $\vec{y}$ and $\vec{x}$:

$$f : \vec{x} \to \vec{y} \tag{1.27}$$

We now consider ways we can carry out Diffusion which learns this map as well. To do so, let us first consider **Classifier-Guided Diffusion**.

### 1.4.1 Classifier-Guided Diffusion

Classifier-Guided Diffusion was introduced by ([13]). Let our classifier $f_\phi(y \mid \vec{x}_t)$ encode the information about the map from $\vec{x}$ to y. Since we are now incorporating information about y, we are dealing with the joint probability distribution:

$$q\left(y, \vec{x}_t\right) = q\left(y \mid \vec{x}_t\right) q\left(\vec{x}_t\right)$$

The associated score-function $s = \nabla_{\vec{x}_t} \log q(\vec{x})$ is the following:

$$\nabla_{\vec{x}_t} \log q\left(y, \vec{x}_t\right) = \nabla_{\vec{x}_t} \log q\left(\vec{x}_t\right) + \nabla_{\vec{x}_t} \log q\left(y \mid \vec{x}_t\right)$$
$$\nabla_{\vec{x}_t} \log q\left(y, \vec{x}_t\right) \approx -\frac{1}{\sqrt{1-\bar{x}_t}} \varepsilon_\theta\left(\vec{x}_t, t\right) + \nabla_{\vec{x}_t} \log q\left(y \mid \vec{x}_t\right)$$

The map $q\left(y \mid \vec{x}_t\right)$ is approximated by our classifier, and thus we write this as following:

$$\nabla_{\vec{x}_t} \log q\left(y, \vec{x}_t\right) \approx -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta\left(\vec{x}_t, t\right) + \nabla_{\vec{x}_t} \log f_\phi\left(y \mid \vec{x}_t, t\right)$$

Factoring out $\frac{1}{\sqrt{1-\bar{\sigma}_t}}$ :

$$\nabla_{\vec{x}_t} \log q\left(y, \vec{x}_t\right) \approx -\frac{1}{\sqrt{1-\bar{\sigma}_t}} \left(\varepsilon_\theta\left(\vec{x}_t, t\right) - \sqrt{1-\lambda} \nabla_{\vec{x}_t} \log f_\phi\left(y \mid \vec{x}_t, t\right)\right) \tag{1.28}$$

Thereby a new classifier-guided predictor $\overline{\varepsilon_\theta}$ takes the following form:

$$\overline{\varepsilon_\theta}\left(\vec{x}_t\right) = \varepsilon_\theta\left(\vec{x}_t\right) - \sqrt{1-\lambda} \nabla_{\vec{x}_t} \log f_\phi\left(y \mid \vec{x}_t, t\right) \tag{1.29}$$

To control the strength of the classifier guidance, we can add a weight w to the delta part,

$$\overline{\varepsilon_\theta}\left(\vec{x}_t\right) = \varepsilon_\theta\left(\vec{x}_t\right) - \sqrt{1-\lambda} w \nabla_{\vec{x}_t} \log f_\phi\left(y \mid \vec{x}_t, t\right) \tag{1.30}$$

Therefore the modified reverse process is given by equation (1.30), where $\varepsilon_\theta\left(\vec{x}_t, t\right)$ was the noise predicted without any guidance and $\overline{\varepsilon_\theta}\left(\vec{x}_t, t\right)$ represents the scaled noise.

Unfortunately there are a few snags that make conditioned classification impractical.Firstly, Classifier guidance complicates the diffusion model training pipeline because it requires training an extra classifier, and this classifier must be trained on noisy data so it is *generally not possible to plug in a pre-trained classifier*. Furthermore, because classifier guidance mixes a score estimate with a classifier gradient during sampling, classifier-guided diffusion sampling can be interpreted as *attempting to confuse an image classifier with a gradient-based adversarial attack*. Even if we have a noise-robust classifier on hand, classifier guidance is inherently limited in its effectiveness: *most of the information in the input $\vec{x}_t$ is not relevant to predicting $\vec{y}_t$*, and as a result, taking the gradient of the classifier w.r.t. its input can yield arbitrary (and even adversarial) directions in input space.

### 1.4.2 Classifier Free Guidance:

Classifier-Free diffusion guidance was introduced by ([21]). In classifier free guidance works, we will use the same model to learn the map $f : \vec{x} \to \vec{y}$ rather than using an external classifier $f_\phi(y \mid \vec{x}_t, t)$ to obtain estimates of it. Let the unconditional denoising diffusion model $q_\theta(\vec{x})$ be parametrized through a score estimator $\varepsilon_\theta(\vec{x}, t)$ and the conditional model $q_\theta(\vec{x} \mid y)$ parametrized through $\varepsilon_\theta(\vec{x}, t, y)$. These two models can be learned via a single neural network. Precisely, a conditional diffusion model $q_\theta(\vec{x} \mid y)$ is trained on paired data $(\vec{x}, y)$ where the conditioning information y gets discarded periodically at random such that the model knows how to generate images unconditionally as well, i.e.

$$\varepsilon_\theta(\vec{x}, t) = \varepsilon_\theta(\vec{x}, t, y = \emptyset) \tag{1.31}$$

The gradient of an implicit classifier can be represented with conditional and unconditional score estimators. Once plugged into the classifier-guided modified score, the score contains no dependency on a separate classifier. To understand this, consider the Bayes Rule;

$$q(y \mid \vec{x}_t) = \frac{q(\vec{x}_t \mid y)\, q(y)}{q(\vec{x}_t)} \tag{1.32}$$

Taking the logarithm and then evaluating the score function:

$$\log q(y \mid \vec{x}_t) = \log q(\vec{x}_t \mid y) + \log q(y) - \log q(\vec{x}_t) \tag{1.33}$$

$$\nabla_{\vec{x}_t} \log q(y \mid \vec{x}_t) \approx \nabla_{\vec{x}_i} \log q(\vec{x}_t \mid y) - \nabla_{\vec{x}_i} \log q(\vec{x}_t) \tag{1.34}$$

$$\nabla_{\vec{x}_i} \log q(y|\vec{x}_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \left(\varepsilon_\theta(\vec{x}_t, t, y) - \varepsilon_\theta(\vec{x}_t, t)\right) \tag{1.35}$$

To control the strength of the classifier guidance, we introduce a weighted sum:

$$\begin{aligned}
\overline{\varepsilon_\theta}(\vec{x}_t, t, y) &= \overline{\varepsilon_\theta}(\vec{x}_t, t, y) - \sqrt{1-\lambda} w \nabla_{\vec{x}_t} \log q(y \mid \vec{x}_t) \\
\overline{\varepsilon_\theta}(\vec{x}_t, t, y) &= \overline{\varepsilon_\theta}(\vec{x}_t, t, y) - w\left(\overline{\varepsilon_\theta}(\vec{x}_t, t, y) - \varepsilon_\theta(\vec{x}_t, t)\right) \\
\overline{\varepsilon_\theta}(\vec{x}_t, t, y) &= (w+1)\overline{\varepsilon_\theta}(\vec{x}_t, t, y) - w\overline{\varepsilon_\theta}(\vec{x}_t, t)
\end{aligned} \tag{1.36}$$

The superior performance of this approach compared to classifier guidance can be attributed to the construction of the "classifier" using a generative model. As discussed, traditional classifiers often exploit shortcuts, selectively ignoring significant parts of the input while still achieving competitive classification results. In contrast, generative models are required to fully engage with the input which results in gradients that are

significantly more robust. Moreover, this method streamlines the training process by requiring only a single generative model. It is noteworthy that the adoption of classifier-free guidance occurred rapidly after its introduction. OpenAI's GLIDE model, for instance, demonstrated its effectiveness shortly after the concept's publication.

## 1.5 Stable Diffusion

Also known as Latent Diffusion, Stable diffusion was introduced in the paper ([31]) by R Rombach et al. For the rest of this section, all of the dimensions that we would state are typically those found when working with the Stable Diffusion model which is publicly accessible through Hugging Face Library.

Let us denote the original image as $x_0$ where the image typically has the dimensions $x_0 \in \mathbb{R}^{3x512x512}$. Instead of working directly in the high-dimensional data space, Latent Diffusion models operate in a lower-dimensional latent space. The idea is to encode the high-dimensional data into a compact representation using an encoder, perform diffusion in this latent space, and then decode it back to the data space. More formally, given a data sample $\vec{x}_0$, Stable Diffusion encodes it into a latent space $\vec{z}_0$ using an encoder E :

$$\vec{z}_0 = E\left(\vec{x}_0\right) \tag{1.37}$$

Where $z_0 \in \mathbb{R}^{4x64x64}$. The encoder E utilized in the Stable Diffusion pipeline is **VQ-VAE**'s encoder. The forward diffusion process in latent space are a series of transitions from $\vec{z}_0$ to $\vec{z}_T$ by gradually adding noise:

$$\vec{z}_t = \sqrt{\bar{\alpha}_t}\,\vec{z}_0 + \sqrt{1-\bar{\alpha}_t}\,\vec{\epsilon}_t, \quad \vec{\epsilon}_t \sim \mathcal{N}(0, \mathbf{I}), \tag{1.38}$$

where $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ represents the cumulative signal retention up to step $t$.

The reverse process is governed by the Kernel:

$$p_\theta(\vec{z}_{t-1} \mid \vec{z}_t, \vec{c}) = \mathcal{N}(\vec{z}_{t-1}; \mu_\theta(\vec{z}_t, t, \vec{c}), \Sigma_\theta(\vec{z}_t, t, \vec{c})), \tag{1.39}$$

Where $\vec{c}$ is the text we are conditioning on.

**Understanding the conditioning text $\vec{c}$ in Stable Diffusion II**

Suppose you type the prompt $\mathcal{P}$ = "A Cat and a Dog". Each of the word would constitute a vector $e_1, ...e_n$ where $e_i \in \mathbb{R}^{1x768}$. Automatically, an embedding for **SOS** and **EOS** would be generated. Therefore, even though there are a total

of 5 words in our prompt, with the inclusion of SOS and EOS tag, these would come to 7. In fact empirically it is found that SOS has consistent high attention as indicated by works of ([9]).
Stable Diffusion II operate with a fixed number of tokens in each sequence. This results in the following fixed sequence size:

$$\vec{c} = e_1, ..e_n \in \mathbb{R}^{77x768} \tag{1.40}$$

The reason for this fixed token length is that during text-conditioning, the attention mechanism is applied which requires computing cross-attention from the key vector. Allowing the embedding sequence length to vary would mean that each text prompt could have a different number of tokens which would result in variable shapes for each example in the batch. Such variability would complicate parallel processing.
From the discussion above, we see that even when the input consists of a single word such as 'dog' with an embedding dimension of $[1, 768]$, the sequence is padded to maintain the fixed length of $[77, 768]$. The remaining tokens are zero-padded (see figure below)

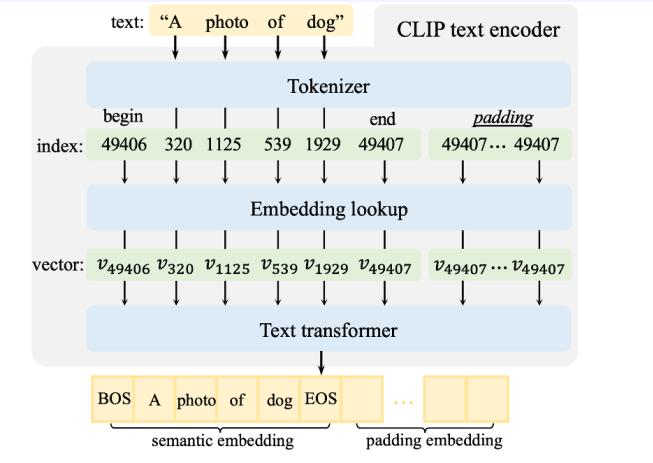


Figure 1: Understanding the structure of embeddings in SD2, figure taken from [40]

Work by [40] highlight the correlations that exist between each embedding due to *causal mask* and the *omission of the padding mask*.

1. **Presence of Causal Masks**: Causal Masks exist in the *self-attention layer of the CLIP encoder*. Due to these, the token $e_i$ can only attend to

the token that precedes it $e_1, e_2, \ldots, e_{i-1}$ but cannot attend to tokens that follow. This encodes means that earlier words establish broader context and subsequent words refining details. Note that these causal masks can explain many of the troubles that SD2 generations encounter.If causal masking overly prioritizes earlier tokens later tokens might not be given sufficient weight. For example in the prompt "A dog and a cat on a couch", the model may focus more on "dog" and neglect "cat" if causal masking weakens attention on tokens toward the end.

2. **Lack of Padding Mask**: In standard transformer architectures, a padding mask is employed to ignore padding tokens—additional tokens added to shorter sequences to align them with the maximum sequence length within a batch. However, in diffusion models, the padding mask is omitted. This omission means that padding tokens are included in attention calculations and can interact with semantic tokens (tokens containing meaningful content). Consequently, padding tokens gain information from their neighboring semantic tokens. This interaction enables padding tokens to acquire information that is not semantically significant but is instead related to style or background information, an observation that ([40]) utilize for edits.

To estimate the reverse kernel in Equation (1.39), the following loss-function is employed:

$$\mathcal{L} = \mathbb{E}_{t,\vec{z}_0,\vec{\epsilon}_t}\left[\left\|\vec{\epsilon}_t - \epsilon_\theta(\vec{z}_t, t, \vec{c})\right\|_2^2\right], \tag{1.41}$$

where $\vec{\epsilon}_t \sim \mathcal{N}(0, \mathbf{I})$ is the true noise added in the forward process and $\epsilon_\theta(\vec{z}_t, t, \vec{c})$ is the neural network's predicted noise conditioned on $\vec{z}_t$, $t$, and the conditioning vector $\vec{c}$. In SD2 pipeline, the neural network that provides estimates $\epsilon_\theta(\vec{z}_t, t, \vec{c})$ is a **U-NeT**. For SD3, the estimates are provided by **DiT**. The neural network gradually denoise $\vec{z_T}$ back to the latent code $\vec{z}_0$. Once the reverse diffusion process in the latent space yields a latent code $\vec{y}_0$, the decoder D is used to map this back to the data space:

$$\vec{x}_0 = D\left(\vec{z}_0\right) \tag{1.42}$$

Where the Decoder is VQ-VAE's decoder.

### 1.5.1 U-NeT of Stable Diffusion

For a single denoising step, a diffusion model must learn to predict $\vec{z}_{t-1}$ given $\vec{z}_t$. Since both $\vec{z}_{t-1}, \vec{z}_t \in \mathbb{R}^d$, we are looking for architectures that preserve the original dimensions of the image. We know that maintaining the entire dimension $d$ throughout the architecture is ineffective and a down sampling layer would reduce the number of parameters of our architecture. An architecture that naturally fits this description is the U-NeT as introduced in the work [32]

Let $\vec{z_t} \in \mathbb{R}^{64x64x4}$ represent the noise latent at time step t. The latent is passed through the U-NeT alongside the conditioning prompt $\vec{c}$. Let us denote the features in the U-NeT as $f_l^t$ where $l$ represent the layer of the U-NeT and $t$ represent the Diffusion timestep. Then, the evolution of $\vec{z_t}$ through the U-NeT is:

1. **Input Features**: The input features are represented as $f_0^t \in \mathbb{R}^{4\times64\times64}$. An initial convolution layer transforms the features to $f_1^t \in \mathbb{R}^{320\times64\times64}$

2. **Downsampling Block**: Consist of 4 Downsampling blocks that progressively reduces the spatial dimension of the initial latent. In Down Block 1, the features are downsampled to $f_2^t \in \mathbb{R}^{320\times32\times32}$. Down Block 2 downsamples this to $f_3^t \in \mathbb{R}^{640\times16\times16}$. Similarly, Down Block 3 outputs features as $f_4^t \in \mathbb{R}^{1280\times8\times8}$. In Down Block 4, the spatial resolution remains constant at $8 \times 8$, and the features are maintained at $f_5^t \in \mathbb{R}^{1280\times8\times8}$.

3. **Midle Block**: This serves as the bottleneck. We have $f_6^t \in \mathbb{R}^{1280\times8\times8}$,

4. **Upsampling Block**: This progressively increases the size of the dimension until it is back to the original size. In Up Block 1, the features are upsampled to $f_7^t \in \mathbb{R}^{1280\times16\times16}$. The Up Block 2 increases the resolution to $f_8^t \in \mathbb{R}^{640\times32\times32}$.. Finally, in Up Block 3 we have $f_9^t \in \mathbb{R}^{320\times64\times64}$.

The output from the U-Net is the predicted noise at timestep t:

$$\vec{\epsilon_t}(\vec{z}_t, t, \vec{c}) = f_{10}^t \tag{1.43}$$

Subsequently, we introduce an empty prompt $\mathcal{P}$=" " and condition the U-NeT on $\vec{z}_t$ and the empty prompt's embedding $\vec{c}$ to obtain the estimate $\epsilon_\theta(\vec{z}_t, t)$. The estimates $\vec{\epsilon}_\theta(\vec{z}_t, \vec{c}, t)$ and $\epsilon_\theta(\vec{z}_t, t)$ are then utilized to perform classifier-free guidance as encapsulated by (1.36). The guided noise estimate becomes:

$$\vec{\epsilon}_{\text{guided}} = \vec{\epsilon}_\theta(\vec{z}_t, t) + w \cdot (\vec{\epsilon}_\theta(\vec{z}_t, \vec{c}, t) - \vec{\epsilon}_\theta(\vec{z}_t, t)) \,, \tag{1.44}$$

where $w \geq 1$ is the guidance scale that controls the strength of the conditioning. A higher $w$ increases adherence to the prompt but may reduce diversity in the output. The guided noise estimate $\vec{\epsilon}_{\text{guided}}$ is then used in the denoising step to refine the latent representation:

$$\vec{z}_{t-1} = \vec{z}_t - \alpha_t \cdot \vec{\epsilon}_{\text{guided}}, \tag{1.45}$$

where $\alpha_t$ is a step size parameter that depends on the diffusion process.

Until now, we have not talked about how exactly the conditioning embedding $\vec{c} \in \mathbb{R}^{77x768}$ interacts with the spatial features $f_l^t$ in the U-NeT. This is mediated through **Cross-Attention** as shown in the figure below.

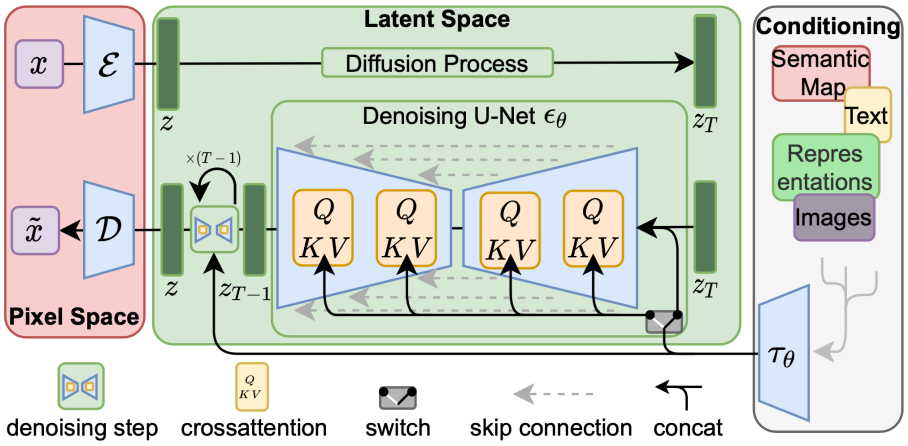


Figure 2: The architecture of Stable Diffusion 2, figure taken from [31]

At each layer $l$, the conditioning vector $\vec{c}$ is projected by a projection layer $\tau_\theta$, which maps $\vec{c}$ to an $d_{embed}$ that agrees with the dimension of spatial features:

$$\tau_\theta(\vec{c}) \in \mathbb{R}^{Mxd_{embed}} \tag{1.46}$$

Here, M = 77 is the length of the sequence and $d_{embed}$ is the dimensionality of the embedded vector which depends upon the layer. The cross-attention is computed as:

$$\text{Cross-Attention (Q, K, V)} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{1.47}$$

Where the Query $Q \in \mathbb{R}^{(hxw)xd_{embed}}$ comes from the **latent features** $\vec{f}_l^t$ of the U-Net and Keys $K$ and Values $V$ comes from the **condition input** $\vec{c}$ such that $K, V \in \mathbb{R}^{Mxd_{embed}}$.

The Query, Value, and Keys are derived from:

$$\mathrm{Q} = W_Q^{(i)} \phi_i \left( \vec{f}_t \right), \qquad \mathrm{K} = W_K^{(i)} \tau_\theta(c), \qquad \mathrm{V} = W_V^{(i)} \tau_\theta(c) \tag{1.48}$$

Where $\phi_i \left( \vec{f}_t \right)$ is the projection layer. To sum up, at each layer $l$ , the model computes the Queries from the U-Net's current feature map $\phi_i \left( \vec{z}_t \right)$ and the Keys K and Value V from the encoded conditioning input $\tau_\theta(c)$

In addition to cross-attention, **self-attention** plays a crucial role at each layer $l$ of the network. Self-attention enables the model to capture spatial dependencies within the latent features $\vec{f}_l^t$. This ensures that information is shared across different parts of the image latent. The self-attention mechanism assumes the form:

$$\text{Self-Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right) V \tag{1.49}$$

Where the Queries $Q$, Keys $K$, and Values $V$ are *all derived from the same source, namely the spatial features* $\vec{f}_l^t$ *at layer* $l$. At each layer $l$, the spatial features are projected into the Query, Key, and Value spaces:

$$Q = W_Q \phi(\vec{f}_l^t), \quad K = W_K \phi(\vec{f}_l^t), \quad V = W_V \phi(\vec{f}_l^t) \tag{1.50}$$

Where $W_Q$, $W_K$, $W_V$ are learnable weight matrices for the Query, Key, and Value projections. Self-attention allows the model to capture long-range dependencies by enabling information from distant spatial regions within the latent features to influence each other.

Table 1: Attention Structure in Stable Diffusion, where $B$ = Batch Size

| **Layer** | **Resolution (X)** | **Channel Dim. (C)** | **Self-Attention Matrix** | **Cross-Attention Matrix** |
|---|---|---|---|---|
| **Before Down Layer 1** | $64 \times 64$ | 320 | $[B \times 4096 \times 4096]$ | $[B \times 4096 \times 77]$ |
| **After Down Layer 1** | $32 \times 32$ | 320 | $[B \times 1024 \times 1024]$ | $[B \times 1024 \times 77]$ |
| **Down Layer 2** | $16 \times 16$ | 640 | $[B \times 256 \times 256]$ | $[B \times 256 \times 77]$ |
| **Down Layer 3** | $8 \times 8$ | 1280 | None | None |
| **Mid Layer** | $8 \times 8$ | 1280 | $[B \times 64 \times 64]$ | $[B \times 64 \times 77]$ |
| **Up Layer 1** | $16 \times 16$ | 640 | $[B \times 256 \times 256]$ | $[B \times 256 \times 77]$ |
| **Up Layer 2** | $32 \times 32$ | 320 | $[B \times 1024 \times 1024]$ | $[B \times 1024 \times 77]$ |
| **Up Layer 3** | $64 \times 64$ | 320 | $[B \times 4096 \times 4096]$ | $[B \times 4096 \times 77]$ |

# 2 Representation learning

## 2.1 Overview

**Representation learning** [6] refers to the process of discovering representations of data that simplify the extraction of meaningful information. In probabilistic approaches, effective representations often reflect the posterior distribution of the key explanatory factors underlying the observed data. Additionally, a useful representation serves as valuable input for supervised prediction tasks. Among the various techniques for learning representations, **deep learning methods** involve the sequential application of multiple non-linear transformations aimed at producing increasingly abstract — and ultimately more effective — data representations.

More formally, if the input data $x$ lives in a high-dimensional space $\mathcal{X}$, then representation learning seeks to map $x$ into a new space $\mathcal{Z}$ where the representation $z$ of the data is more useful for downstream tasks such as classification and regression. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Z}$, where:

$$z = f(x; \theta), \tag{2.1}$$

where $x$ is the raw input data (e.g., pixels, words, sensor values), $z$ is the learned representation and $\theta$ are the parameters of the model.

Traditionally, deep learning involves learning representations are learned through neural networks. Each layer of a neural network transforms the input $x$ into a new representation $h$ using nonlinear transformations. For a layer $l$, the transformation is:

$$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)}), \tag{2.2}$$

where, $h^{(l)}$ is the representation at layer $l$ and $\sigma$ is a nonlinear activation function. The transformations are compositional, and thus the learning is hierarchical:

$$z = f_L(f_{L-1}(\dots f_1(x; \theta_1) \dots; \theta_{L-1}); \theta_L), \tag{2.3}$$

where each $f_l$ represents a layer transformation with its own parameters $\theta_l$.

The representations $z$ are learned by optimizing a loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}} \left[ \ell(f(x; \theta), y) \right], \tag{2.4}$$

where $\ell$ is the task-specific loss function (e.g., cross-entropy for classification), $f(x; \theta)$

produces predictions or embeddings based on learned representations and $y$ is the ground truth label. During training, the network adjusts $\theta$ such that the learned representations $z$ are optimal for the task.

For **generative models**, representation learning focuses on discovering latent factors $z$ that generate the observed data $x$:

$$p(x, z) = p(x \mid z)p(z), \tag{2.5}$$

where $p(x \mid z)$ represents the conditional likelihood of the data given latent variables and $p(z)$ is the prior distribution over the latent variables.

A good representation $z$ efficiently encodes the structure of $x$ while disentangling independent factors of variation. For inference, we estimate the posterior $p(z \mid x)$, which often requires approximations in complex settings. The posterior distribution $p(z \mid x)$ determines the latent factors $z$ given observed data $x$. Using Bayes' theorem, we get:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}, \tag{2.6}$$

where $p(x)$ is the marginal likelihood or evidence, defined as:

$$p(x) = \int p(x \mid z)p(z)\, dz. \tag{2.7}$$

This integral sums over all possible latent variables $z$ to compute the likelihood of $x$. The posterior $p(z \mid x)$ combines the prior $p(z)$ and likelihood $p(x \mid z)$, balancing prior assumptions with evidence from the data. However, for complex models, the integral $\int p(x \mid z)p(z)\, dz$ is intractable, making direct computation of $p(z \mid x)$ infeasible.

The goal of representation learning is to optimize the model parameters such that:

1. The latent variables $z$ capture the underlying structure of the data $x$.

2. The posterior $p(z \mid x)$ disentangles independent factors of variation.

The traditional way in order to learn the model parameters (e.g., in $p(x \mid z)$ and $p(z)$) is to maximize the marginal likelihood of the data $x$:

$$\log p(x) = \log \int p(x \mid z)p(z)\, dz. \tag{2.8}$$

Direct maximization is often difficult because of the integral over $z$. The most common solutions are **variational inference** in which you approximate $p(z \mid x)$ with

a simpler distribution $q(z \mid x)$ and **Monte Carlo Methods** which uses sampling techniques to approximate the integral.

## 2.2 Manifolds and Representation

**Manifold Hypothesis**

An elegant way to understand a representation of a model is through manifold hypothesis. The manifold hypothesis states that real-world high-dimensional data lies on a low-dimensional manifold embedded in the high-dimensional space. Mathematically, let the data be $x \in \mathbb{R}^D$ where D is large. Then, the data can be well-approximated by a much lower-dimensional space a manifold $\mathcal{M} \subseteq \mathbb{R}^D$ where $\dim(\mathcal{M}) = d \ll D$.

Thus, for any "realistic" input x , there exists a latent representation h such that:

$$x \approx g_\phi(h) \text{ where h} \in \mathbb{R}^D, \dim(\mathcal{M}) = \text{d} \tag{2.9}$$

When we say that an input $x$ ' is "off the manifold", it means $x$ ' does not lie close to the learned manifold $\mathcal{M}$. Consequently, there is no corresponding latent representation h $\in \mathbb{R}^d$ such that the condition $x' \approx g_\phi(h)$ is satisfied.

Most machine learning algorithms assume inputs are sampled from the data manifold $\mathcal{M}$ where they were trained. If the input lies far from $\mathcal{M}$, unusual or unexpected behavior can occur. Formally, a Manifold is a topological space that locally resembles Euclidean space near each point. Globally, the manifold can have a complex shape or curvature. For example, although a 2D sphere $S^2 \subset \mathbb{R}^3$ is not globally flat, but each region on the sphere resembles $\mathbb{R}^2$.

The criterion that a manifold must locally resemble a Euclidean Space is to ensure that the rules of calculus carry over. In particular, it ensures:

1. **Existence of Tangent Spaces** At each point p, the manifold locally behaves like $\mathbb{R}^d$, so we can define a tangent space to it (much like tangent spaces define directional derivatives in 2D)

2. **Criterion of Smooth Maps** Functions defined on manifolds can be analyzed locally using smooth coordinate charts.

Manifolds are characterized by **tangent spaces**. They specify how x can change while staying on manifold. At a point **x** on a d-dimensional manifold, the tangent

plane is given by basis vectors that span the local directions of variation allowed on the manifold. More formally, if $\mathcal{N}$ is defined implicitly as a smooth mapping x $= g(h)$, the Jacobian of g, $J_g$ describes how the manifold is locally oritented:

$$J_g(h) = \frac{\partial g(h)}{\partial h} \in \mathbb{R}^{D \times d} \tag{2.10}$$

For example, for a sphere, the smooth mapping $g : \mathbb{R}^2 \to \mathbb{R}^3$ is:

$$g(h) = g(\theta, \phi) = \begin{bmatrix} r\sin(\phi)\cos(\theta) \\ r\sin(\phi)\sin(\theta) \\ r\cos(\phi) \end{bmatrix}$$

The Jacobian matrix of this is:

$$J_g(\theta, \phi) = \begin{bmatrix} \frac{\partial g_1}{\partial \theta} & \frac{\partial g_1}{\partial \phi} \\ \frac{\partial g_2}{\partial \theta} & \frac{\partial g_2}{\partial \phi} \\ \frac{\partial g_3}{\partial \theta} & \frac{\partial g_3}{\partial \phi} \end{bmatrix} = \begin{bmatrix} -r\sin(\phi)\sin(\theta) & r\cos(\phi)\cos(\theta) \\ r\sin(\phi)\cos(\theta) & r\cos(\phi)\sin(\theta) \\ 0 & -r\sin(\phi) \end{bmatrix}$$

The columns of the Jacobian matrix are the basis vectors of the tangent space at the point x :

- $\frac{\partial g}{\partial \theta}$ describes the direction of change when $\theta$ varies (moving along the azimuthal angle).
- $\frac{\partial g}{\partial \phi}$ describes the direction of change when $\phi$ varies (moving along the azimuthal angle).

To understand the above in context of example, take the MNIST data. Each data point **x** lies on a low-dimensional manifold. The tangent space at **x** provides the directions of allowable variation (e.g., rotations, translations, or deformations of the digit). The Jacobian of a learned mapping $g : \mathbb{R}^d \to \mathbb{R}^D$ describes these directions. The key insight is that manifolds locally behave like flat planes and the Jacobian provides the link between the low-dimensional intrinsic structure and the high-dimensional ambient space.

Let us now get a sense of **orthogonal spaces** to manifolds. Say that the lower dimensional representation that we are learning is defined by $g : h \to x$ where $g : \mathbb{R}^d \to \mathbb{R}^D$ and $J_g(h) = \frac{\partial g(h)}{\partial h} \in \mathbb{R}^{Dxd}$ spans the tangential space to the manifold. As explained, movements along the tangent space correspond to meaningful variations on the manifold and thus affect h which is the lower-dimensional representation of data. The normal space $N_x\mathcal{M}$ is the ( $D - d$ ) dimensional complement of the tangent

space in $\mathbb{R}^D$. Movements orthogonal to the manifold correspond to deviations that are not part of the manifold and therefore do not affect h. For the case of our sphere, tangential directions (along the surface) correspond to changes in the angles $\theta$ and $\phi$, which are meaningful. The representation g that we would learn would only capture the meaningful changes in $\theta$ and $\phi$ and ignores variations orthogonal to the sphere's surface.

Thus, given an arbitrary point $h \in \mathbb{R}^d$, the data-point x $\in \mathbb{R}^D$ can be decomposed into a tangential component $x_T$ and an orthogonal component $x_\perp$ with respect to the data manifold. The movements along the orthogonal component $x_\perp$ do not capture the manifold that our representation $g$ has learned. Thus, they do not effect h. On the other hand, movements along the tangential component $x_T$ capture the manifold.

More formally, given the Jacobian $J_g(h) = \frac{\partial g(h)}{\partial h}$, our representation must ignore the variations orthogonal to the manifold while not that which is along the manifold:

$$\begin{aligned} \frac{\partial g(h)}{\partial h} \cdot x_\perp &= 0 \\ \frac{\partial g(h)}{\partial h} \cdot x_T &\neq 0 \end{aligned} \tag{2.11}$$

To understand this in context of an example, suppose we have a representation (say, an autoencoder), which learns to capture and encode the variations in the latent representation h for a specific digit image x (e.g, a " 3 "). Now, variations along the manifold are changes that correspond to meaningful variations of the digit. For example, making the stroke thicker or thinner, rotating the digit slightly or changing its style (e.g., curvy vs. angular). These variations stay on the manifold because they still represent a valid "3". On the other hand, variations along the manifold do not correspond to meaningful digits. These include adding random noise to the pixels or perturbing parts of the image in a way that doesn't look like a "3" anymore (e.g., stray dots or blurs). These variations move the image off the manifold.

To understand how examining the representation of models using **manifold learning** can provide us practical insights into their workings, lets take the example of **autoencoders**. We can think of autoencoder as projecting the data into a lower-dimensional representation (latent space) and reconstructing it back to effectively capture the structure of the data manifold.

Let the data be $x \in \mathbb{R}^D$ where D is large. In autoencoders, an encoder is responsible for mapping the input data x to a lower-dimensional representation h : $f_\theta : \mathbb{R}^D \to \mathbb{R}^d$ while a decoder is responsible for reconstructing the input x from its lower dimensional representation h : $g_\phi : \mathbb{R}^d \to \mathbb{R}^D$. The reconstructed input is:

$$x' = g_\phi\left(f_\theta(x)\right) \tag{2.12}$$

The objective of the autoencoder is to minimize the reconstruction error:

$$L_{rec} = \|x - g_\phi\left(f_\theta(x)\right)\|^2 \tag{2.13}$$

The lower-dimensional representation h learned by the encoder is often referred to as the "latent space" or "encoding space." This space captures the essential features of the input data. Now, given that we have been able to capture a lower-dimensional manifold d using autoencoders, there are two types of possible variations with respect to the original data x that the autoencoder can handle.For the tangential Variations (On the Manifold), the autoencoder learns to capture and encode these variations in the latent representation h. For example, if you input an image of "3" with slightly thicker strokes, the latent representation $h$ will reflect this meaningful variation. The reconstruction $f(x)$ will match the input closely because it stays on the manifold. If noise or irrelevant perturbations are added to the image (e.g., a speck of noise in the background), the autoencoder projects the input back onto the manifold. This means the autoencoder "corrects" the image, ignoring the noise, and outputs a clean version of the "3."

The thing is, equation (2.13) focuses solely on minimizing the reconstruction error without any attempt at capturing the underlying patterns of the data. The autoencoder learns to reconstruct not just the manifold structure, but also the irrelevant directions (orthogonal to the manifold). This means that the reconstruction error is minimized even for noisy or off-manifold points. The latent representation h = $f_\theta(x)$ becomes sensitive to variations orthogonal to the manifold. As a result without regularization, the latent representation h is not constrainted to be smooth or aligned with the intrinsic manifold directions.

We can formalize this in context of the Jacabian of the manifold. Let $x_{clean}$ be the clean input and $x_{\text{noisy}} = x_{\text{clean}} + \delta x$ the noisy input. If the encoder $f_\theta$ is sensitive to small changes in x , then a small amount of noise causes the latent representation h to change drastically. The noisy input is mapped far from the clean input in the latent space, even though they look visually similar. Now, if the encoder is sensitive to small changes in x , then the norm $\|J_g(h)\|$ of the jacobian is large.

To mitigate the problem of autoencoders overfitting and not capturing the data manifold, Contractive Autoencoders (CAE) penalize the Jacobian norm in the loss function:

$$L_{CAE} = \|x - g_\phi\left(f_\theta(x)\right)\|^2 + \lambda \left\|J_f(x)\right\|_F^2 \tag{2.14}$$

The Jacobian penalty ensures that small input perturbations cause minimal changes in the latent representation. More formally, a small neighborhood of points around an input $x$ in the input space $\mathbb{R}^D$ is mapped to a smaller neighborhood in the latent space or output space $\mathbb{R}^d$. This causes the input space to "contract" or "shrink" locally into a smaller, smoother region in the latent space.

The sensitivity is measured by the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input:

$$\|J_f(\vec{x})\| = \sum_{ij}\left(\frac{\partial h_j(\vec{x})}{\partial x_i}\right)^2 \tag{2.15}$$

Where $h_j$ is one unit output in the compressed code $\vec{z} = f(x)$. This penalty term is the sum of squares of all partial derivatives of the learned encoding with respect to input dimensions. The authors claimed that empirically this penalty was found to carve a representation that corresponds to a lower-dimensional non-linear manifold, while staying more invariant to majority directions orthogonal to the manifold.

Another solution to avoid overfitting was introduced in [37]. They proposed a modification to the basic autoencoder in which the input is partially corrupted by adding noises to or masking some values of the input vector in a stochastic manner,

$$\tilde{x} \sim q_D(\tilde{x} \mid \vec{x}) \tag{2.16}$$

Then the model is trained to recover the original input:

$$\begin{gathered}\tilde{x}^{(i)} \sim q_D\left(\tilde{x}^{(i)} \mid \vec{x}^{(i)}\right) \\ L_{\mathrm{DAE}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}\left(\vec{x}^{(i)} - g_\phi\left(f_\theta\left(x^{(i)}\right)\right)\right)^2\end{gathered} \tag{2.17}$$

Where $\mathcal{M}_D$ defines the mapping from the true data samples to the noisy or corrupted ones. For each input, a fixed number $v$ d of components are chosen at random, and their value is forced to 0 , while the others are left untouched. All information about the chosen components is thus removed from that particuler input pattern, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks".

The authors explain why denoising autoencoders learn a more effective representation than standard autoencoders by invoking the **manifold learning** perspective: a corrupted sample lies farther from the data manifold compared to uncorrupted samples. Consequently, the stochastic mapping from the corrupted input $\tilde{x}$ to the reconstructed data $x'$ must learn to make larger adjustments to bring the corrupted input closer to

the manifold. In the limit, the operator should map even distant points to a small region near the manifold.

Finally, let us recall how we can think of the Diffusion process in terms of *manifold learning*. Given the manifold hypothesis, we assume that the posterior distribution $p_{data}(x)$ is not uniformly distributed across the full space and instead is concentrated within a structure subspace.

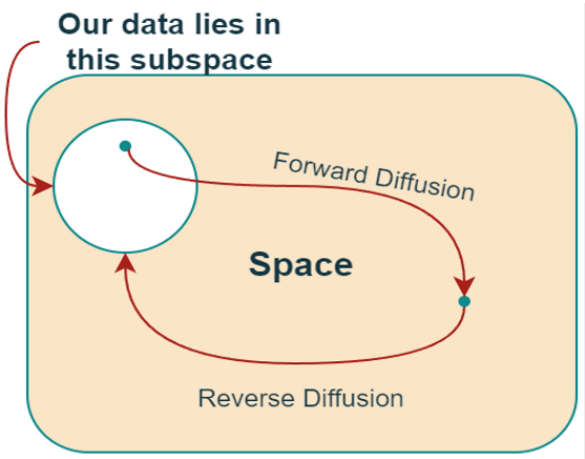


Figure 3: Diffusion as capturing the manifold of the data, Figure taken from a Blogspot

In the figure, the subspace (white circle) represents the lower-dimensional manifold where the data distribution $p_{\text{data}}$ resides while the ambient space (beige background) represents the full high-dimensional space. At the starting point $x_0$, the data is highly structured and confined to the manifold. The forward diffusion process destroys the manifold's structure, progressively injecting noise into the data representation. This eventually resulting in a representation $x_T$ that is isotropic and unstructured. The reverse diffusion process inverts this destructuring process and recovers the structured data distribution confined to the lower-dimensional manifold.

## 2.3 Disentangled Representations

In the 2.2, we saw how manifolds provide a natural way to think about representations of models in Deep Learning. In this section, we are going to be talking about an equally important criterion about good representations: namely, they are **disentangled** where each latent variable captures one factor of variation.

To introduce a motivating example, lets think of GANs. These are generative models consisting of two neural networks, a generator $G$ and a discriminator $D$, both of which are trained simultaneously in a minimax game. The generator $G$ maps random noise $z$ to data space, aiming to produce realistic samples, while the discriminator $D$

tries to distinguish between real data (from the dataset) and fake data (generated by $G$). The objective is to find a Nash equilibrium where $G$ generates data indistinguishable from real data, and $D$ cannot reliably tell real from fake. The GAN loss function is:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim P_{\text{noise}}}\left[\log\left(1 - D(G(z))\right)\right] \tag{2.18}$$

In the normal setting, the input noise $z$ is transformed into an image $I$ in a way that does not guarantee that factors of variations will be disentangled. For example, changing $z$ might modify the face's pose, hairstyle, and lighting simultaneously rather than isolating one attribute.

Enter InfoGANs [10]. These models aim to learn a disentangled and interpretable latent variable in an unsupervised manner. To carry this out, they introduce a structured latent code $c$ which can be *categorical* or *continious* depending upon the nature of the attributes we want to control. By maximizing the **mutual information** $I(c; G(z,c))$ between $c$ and the generated data $G(z)$, InfoGAN ensures that $c$ influence specific features of the data. Thus, the loss function is now represented as (details of how the mutual information on the left-hand side is made tractable is left for the appendix):

$$\min_G \max_D V_{IG}(D,G) = V(D,G) - \lambda I(c; G(z,c)). \tag{2.19}$$

This loss encourages the InfoGAN to use the latent codes $c_i$ in semantically meaningful ways. For example, for the MNIST dataset, InfoGAN might learn the latent code $c$ consisting of one categorical variable $c_1$ which represents digit identity and one continuous variables $c_2$ representing digit rotation which is a continuous transformation. During training, the categorical code $c_1$ encourages $G$ to associate each category of $c_1$ with a specific digit identity. The continuous code allow $G$ to represent continuous variations in the digits. By systematically varying one latent code at a time while keeping others fixed, InfoGAN disentangles these semantic factors: changing $c_1$ switches between digit identities (e.g., from "0" to "9"), changing $c_2$ rotates the digit without altering its identity.

# 3 Image Editing

## 3.1 Overview

Image editing is the process of transforming an input image $\mathbf{I}_{\text{in}} \in \mathbb{R}^{H \times W \times C}$ into an output image $\mathbf{I}_{\text{out}} \in \mathbb{R}^{H' \times W' \times C'}$. This transformation is governed by a function $f$ which applies the desired modifications to the input image based on a set of parameters $\mathbf{P}$. Formally, we can express this as $\mathbf{I}_{\text{out}} = f(\mathbf{I}_{\text{in}}, \mathbf{P})$ where the function $f$ encapsulates editing operations ranging from pixel-level adjustments to semantic modifications.

**Pixel-level editing** involves direct manipulation of the image's pixel values. A prototypical example of pixel-level editing is adjusting the contrast of an image such as applying a grayscale filter to an image. This edit can be expressed as $\mathbf{I}_{\text{out}}(i, j, c) = g(\mathbf{I}_{\text{in}}(i, j, c), \mathbf{P})$ where $g$ modifies the intensity of pixel $(i, j)$ in channel $c$. Another example is a geometric transformation such as scaling or rotation. More advanced forms of image editing involve semantic modifications in which object shapes or relationships is altered. This often requires working in a latent space. A practical example of **Semantic editing** is changing the color of an object in the image or modifying facial expressions in a portrait by manipulating the latent vectors.

Most of the works in Diffusion models usually address the latter form of editing and therefore naturally confront the *representation of these models*. To understand how semantic editing and representation are connected, recall that a key part of learning a good representation is the ability to disentangle different factors of variation in an image such as pose and color. This disentanglement makes it possible to edit specific attributes independently without affecting others. For example, in an image of a car, a disentangled representations can allow edits which change the color of the car while keeping its shape and background as is.

While a good representation aid meaningful edits, the relationship flows both ways with editing serving as an intervention to address shortcomings in the representation of pre-trained diffusion models. The two most common problems that literature in editing of Diffusion models confronts in this regards is **Catastrophic Neglect** and **Attribute Binding**. In Diffusion Models, catastrophic neglect occurs when the model fails to generate one or more specific subjects it was conditioned to train on. On the other hand, Attribute binding refers to the capability of a model to correctly associate specific attributes with the corresponding objects. For example, given the prompt *"P = A blue cat and a brown dog"*, the model generates a brown dog and a blue cat, thereby confusing the properties that the subjects should inherit. Other works

(a) Illustration of Catastrophic Neglect. Given the prompt "A cat and a frog", SD2 only generates a cat.

(b) Illustration of Attribute Binding. Given the prompt "A red chair and a yellow clock", SD2 generates a yellow chair

Ofcourse, there are many other problems. One of the most well-documented limitations of diffusion models lies in their inability to **generate realistic hands** and **coherent text**. The struggle of Diffusion models in generating hands arises from the anatomical complexity of the structure as well as the wide ranges of poses they undergo in natural images. This leads to generations with missing fingers, distorted joints, or unrealistic poses. Similarly, generating text poses a significant challenge due to the need for consistent spatial alignment and font structure. Text-to-image diffusion models struggle with these tasks because their training datasets often lack sufficient examples to capture the nuances that are required for accurate text generation.

(a) Illustration depicting awkward Hand Generation. Given the prompt "A man holding an apple", SD2 generates an unnatural pose for hands.


(b) Illustration of SD2 problem with words. Given the prompt "A board saying: "American Elections, 2024", SD2 generates incomprehensible text

Besides, Diffusion models often fail to capture **spatial relationships** between objects in an image. For instance, when given a prompt like "A cat sitting on a chair," the model generate an image where the cat appears to be floating above the chair or embedded within it which violates spatial logic. On the same note, a prompt such as "A tree beside a lake" produce an image where the tree and lake unnaturally overlap. These shortcomings arise because diffusion models tend to focus on capturing texture and appearance rather than explicitly understanding positional relationships.

In addition, T2I diffusion models struggle with incorporating **commonsense knowledge** which is essential for generating contextually appropriate images. For example, a prompt like "A fish swimming in a bowl, with a bird flying above it" result in an image where the fish is not immersed in water, indicating a lack of understanding of what it means to "swim".

(a) Given the prompt, P = An elephant holding a balloon, the generation fails to capture the relation "holding"

(b) Given the prompt, P =A fish swimming in a bowl, with a bird flying above it, SD2 generations indicates a lack of understanding of commonsensical knowledge

**Different types of Edits**

**Fine-Grained Control**: Fine-grained control leverages signals from Model pipeline to selectively edit specific parts of the image. This allows for precise modifications and ensures that changes are localized and do not disrupt the overall structure or context of the image. Examples include *Prompt-to-Prompt* and *DiffEdit*, *MASA-Ctrl*

**Text-guided modifications**: involve changing attributes of objects in an image such as transforming "a red car" into "a blue car." They can also alter objects entirely like modifying "a dog" into "a robot dog," while maintaining the overall context and composition of the image. An example of Text-guided modification

in Diffusion models is the work *Plug & Play*

**Attribute-Level Editing** focuses on fine-tuning specific characteristics of objects or individuals. For instance, it can adjust facial expressions, change hair color, or modify clothing styles in portraits. Works which target Attribute-level editing in Diffusion models is *Attend and Excite* and *Structure Diffusion*

**Spatial Layout Control**: Spatial layout control enables precise positioning of objects within the image. For example, it can place "a tree on the left" and "a lake on the right" to ensure the layout matches a specific description. In context of Diffusion models, works such as *Spa-Text*, *LLM-Diffusion*, and *Dense T2I Generation* target spatial layout control.

**Appearance Transfer**: Appearance transfer involves modifying the attributes of a target image by incorporating features such as color, texture, or lighting from a reference image while preserving the structural layout or semantic content of the target image.

## 3.2 Fine-Grained Editing

In the overview, we stated how fine-grained editing allows us to selectively edit specific parts of the image. These type of edits allows users to introduce subtle modifications in artwork such as adding details to specific parts. This can be useful in scenarios like advertising where the property ensures that only the desired elements are altered and consistency of the scene is preserved. For example, we might want the color of the chips that are being sold to be of different color than it currently is. Or we might want the pose of the person to be different than it currently is. These fall into the category of fine-grained editing. We next discuss some of the work in literature that addresses these in context of T2I Diffusion Models.

### 3.2.1 Prompt-to-Prompt

Prompt-to-Prompt [20] (P2P) is one of the first **training-free** method that explores the modification of SD2 for edits. The authors leverage the cross-attention layers within the text-conditioned diffusion models to achieve fine-grained editing operations.

Let $\mathcal{J}$ by an image which was generated by a text prompt $\mathcal{P}$. The goal is to edit the input image by providing an additional edited prompt $\mathcal{P}^*$ which results in an edited image $\mathcal{J}^*$. For example, consider an image generated from the prompt $\mathcal{P}^* =$ "my new bicycle". Assume that the user wants to edit the color of the bicycle or replace it with a scooter but preserve the structure of the original image. An intuitive interface for

the user is to directly change the text prompt by further describing the appearance of the bikes or replacing it with another word. However, Diffusion models struggle to retain the features of the previous generation and therefore the need for intervention.

To introduce fine-grained control over the generation process, the authors utilize cross-attention maps which are computed at each time-step $t$ of the Diffusion process and across the layers $l$ of U-NeT. From (1.47), we see that the attention matrix $A_i \in \mathbb{R}^{HWxM}$ where M are the number of tokens, H is the Height and W is the Width of the image. Indexing across the second dimension, and reshaping the first dimension from $HW$ to $HxW$, we obtain:

$$M_{ij} = A_i[j] \in \mathbb{R}^{HxW} \tag{3.1}$$

The matrix $M_{ij}$ defines the weight of the value of the j -th token on the pixel (i,j). Empirically, the authors find that the spatial layout and geometry of the generated image depend on the cross-attention maps. Pixels are more attracted to the words that describe them, e.g., pixels of the bear are correlated with the word "bear". Interestingly, we can see that the structure of the image is already determined in the early steps of the diffusion process.

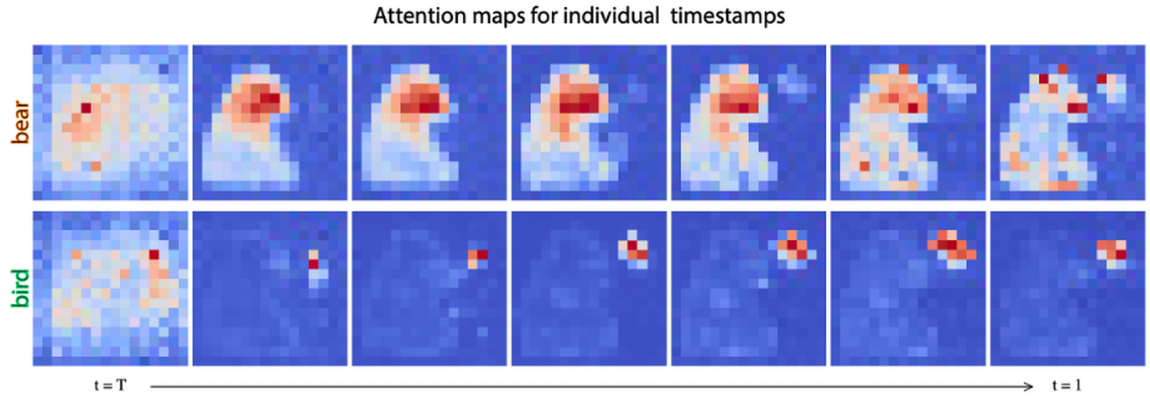


Figure 7: The architecture of Stable Diffusion 2, figure taken from [31]

Since the attention matrix captures the overall composition of an image, we can leverage it to guide edits. Specifically, the attention maps $M$ generated during the creation of an image using the original prompt $\mathcal{P}$ can be injected into a second generation process with a modified prompt $\mathcal{P}^*$. This allows us to produce an edited image $\mathcal{I}^*$ that aligns with the changes described in the new prompt while still preserving the structure of the original input image $\mathcal{J}$.

Let's formalize this. Define $\text{DM}(\vec{z}_t, \mathcal{P}, t, s)$ as a single step $t$ in the diffusion process which outputs the noisy latent $\vec{z}_{t-1}$. Here, $s$ is a random seed that ensures the

generation can be reproduced when the same seed is reused. In this setup:

- $V$ and $M$ denote the value matrix and attention matrix produced by the original prompt $\mathcal{P}$.
- We represent the operation $\text{DM}(\vec{z}_t, \mathcal{P}, t, s)\{M \leftarrow \widehat{M}\}$ as the diffusion step where the attention map $M$ is replaced with a provided map $\widehat{M}$. However, we keep the value matrix $V$ unchanged from the original prompt. $M_t^*$ refers to the attention map generated by the modified prompt $\mathcal{P}^*$.
- Finally, let $\text{Edit}(M_t, M_t^*, t)$ be a general function designed to edit attention maps. It takes as input the attention maps $M_t$ and $M_t^*$ at timestep $t$ during the generation of the original and edited images.

With this foundation, the authors introduce a Prompt-to-Prompt image editing framework to achieve precise and controllable modifications. The algorithm can be described as:

**Algorithm 3** Prompt-to-Prompt Image Editing

**Input:** A source prompt $\mathcal{P}$, a target prompt $\mathcal{P}^*$, and a random seed $s$.
**Output:** A source image $x_{\text{src}}$ and an edited image $x_{\text{dst}}$.

$z_T \sim \mathcal{N}(0, I)$ ▷ Sample a unit Gaussian random variable with random seed $s$
$z_T^* \leftarrow z_T$
**for** $t = T, T-1, \ldots, 1$ **do**
    $z_{t-1}, M_t \leftarrow \text{DM}(z_t, \mathcal{P}, t, s)$ ▷ Denoise source prompt
    $M_t^* \leftarrow \text{DM}(z_t^*, \mathcal{P}^*, t, s)$ ▷ Denoise target prompt
    $\widehat{M}_t \leftarrow \text{Edit}(M_t, M_t^*, t)$ ▷ Perform editing between source and target maps
    $z_{t-1}^* \leftarrow \text{DM}(z_t^*, \mathcal{P}^*, t, s_t)\{M \leftarrow \widehat{M}_t\}$ ▷ Apply edited map to the target
**end for**
**Return** $(z_0, z_0^*)$

According to this algorithm, we first run the diffusion process conditioned on our original prompt $\mathcal{P}$. We denoe this as $DM\left(\vec{z}_t, \mathcal{P}, t, s\right)$. This gives us $\vec{z}_{t-1}$ and $M_t$ where $M_t$ is the attention map associated with original diffusion process. We then copy the noise vector $\vec{z}_t$ and denote that as $\vec{z}_t^*$. We now run the diffusion process on the edited prompt $\mathcal{P}*$ and the copy of the noise vector: $DM\left(\vec{z}_t^*, \mathcal{P}^*, t, s\right)$. This gives us the new modified attention map $M_t^*$.

We then perform Editing in which we take the modified attention map $M_t^*$ and the original attention map $M_t$. This yields a new map $\widehat{M}_t$. Using the edited map $\widehat{M}_t$, the diffusion model predicts the next latent variable $\vec{z}_{t-1}^*$ which is denoted as $\vec{z}_{t-1}^* \leftarrow DM\left(\vec{z}_t^*, \mathcal{P}^*, t, s\right)\left\{M \leftarrow \widehat{M}_t\right\}$

**Defining the edit function** $Edit(M_t$

The following are the three key Edits that Plug & Play defines:

1. **Word Swap**
2. **Adding a new phrase**
3. **Attention Re-weighting**

**Word Swap**: Suppose the user swaps the tokens of the original prompt with others. For example, $\mathcal{P}$ = "a big red bicycle" to $\mathcal{P}^*$ = "a big red car". Now, if we overly constraint the generation of the new image with the attention map $M$ of the original prompt $\mathcal{P}$, the new content might not be fully realized (e.g., the "car" might still look like a "bicycle").

To address this, the authors utilize **attention injection** - a process of controlling the amount and timing of attention map injection during the diffusion process. To balance between preserving the original structure and allowing the new content to appear correctly, the process uses a timestamp parameter ($\tau$). Up the to the timestep $\tau$, the modified attention map $M_t^*$ is used. This allows the new object to start forming based on the new prompt. After $\tau$, the original attention map $M_t$ from the source image is reintroduced, helping to maintain the composition and structure of the original image. Formally, we express this as following:

$$\text{Edit}\left(M_t, M_t^*, t\right) = \begin{cases} M_t^*, & \text{if } t < \tau \\ M_t, & \text{otherwise} \end{cases} \tag{3.2}$$

This means that early in the diffusion process, the new content has more freedom to form using $M_t^*$ but as the process progresses, the original structure is reinforced by switching back to $M_t$. If $\tau$ is set high (near the end of the diffusion process), the final image will look more like the original image but with the new object in place (e.g., a car that might still resemble the original bicycle's shape). If $\tau$ is set low (near the beginning of the process), the new content (car) will have more freedom to adopt its correct shape and appearance, potentially altering the structure more significantly.

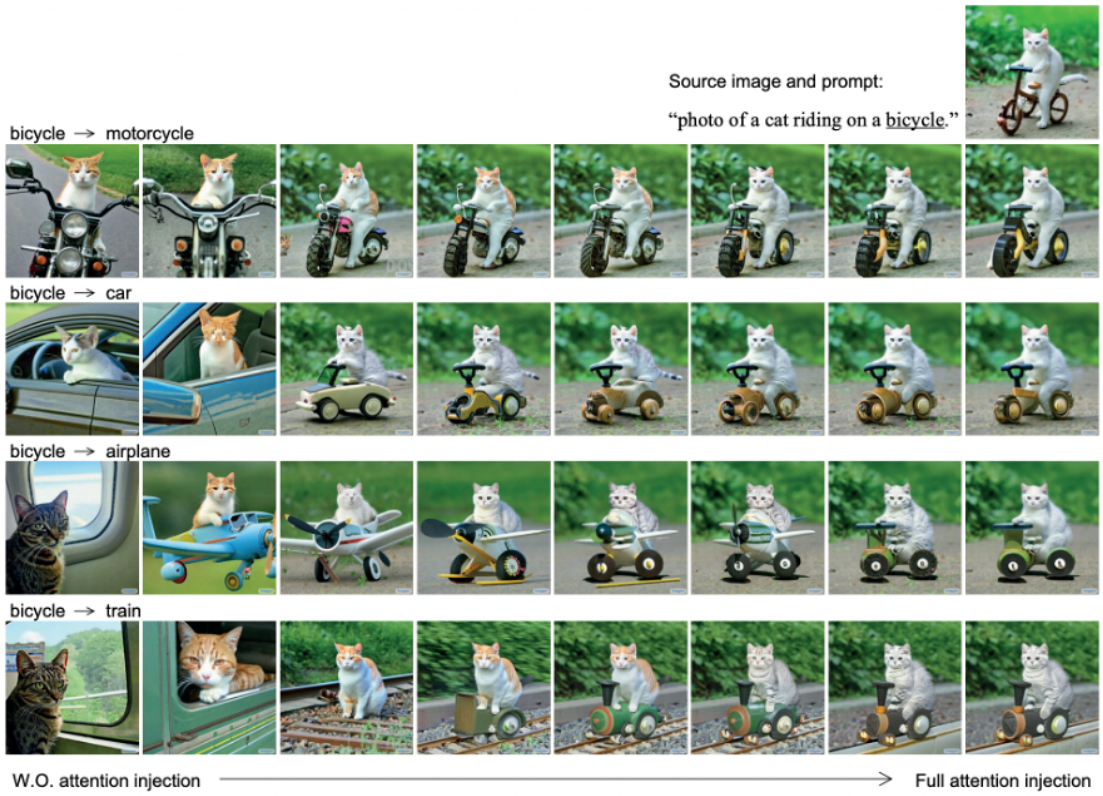


Figure 8: Illustration of attention injection in which the word *bike* is replaced by another word. The figure is taken from [20]

**Adding a New Phrase**: Suppose the user adds new tokens to the prompt, e.g., $\mathcal{P}$ = "a castle next to a river" to $\mathcal{P}^*$ = "children drawing of a castle next to a river". When you add new tokens to a prompt (e.g., changing "a castle next to a river" to "children drawing of a castle next to a river"), the generated image needs to reflect the new stylistic or contextual changes while still retaining the original elements that are common between both prompts. For example, in this case, the commonality is the castle and river).

The **attention alignment** function A is used to map tokens from the modified prompt $\mathcal{P}^*$ back the original prompt $\mathcal{P}$. For each token in the modified prompt $\mathcal{P}^*$, the function A identifies whether this token matches a token in the original prompt $\mathcal{P}$. If it does, A returns the corresponding index from P ; if it doesn't, A return none.

The attention injection is applied selectively based on the alignment function $A$. Formally,

$$\text{Edit}\left(M_t, M_t^*, t\right)_{i,j} = \begin{cases} \left(M_t^*\right)_{i,j}, & \text{if } A(j) = \text{None} \\ \left(M_t\right)_{i,A(j)}, & \text{otherwise} \end{cases} \tag{3.3}$$

In the equation, $i$ corresponds to a pixel in the image, and $j$ corresponds to a token

in the text prompt. To understand this , take $\mathcal{P}$ = "a castle next to a river" and $\mathcal{P}$ * = "children drawing of a castle next to a river". The tokens for these are, respectively:

$$\mathcal{P} = [\text{"a"}, \text{"castle"}, \text{"next"}, \text{"to"}, \text{"a"}, \text{"river"}]$$
$$\mathcal{P}^* = [\text{"children"}, \text{"drawing"}, \text{"of"}, \text{"a"}, \text{"castle"}, \text{"next"}, \text{"to"}, \text{"a"}, \text{"river"}]$$

For "children", "drawing", "and", "of", $A(j) = None$. For "a," "castle," "next," "to," and "river," A(j) maps to the corresponding indices in the original prompt $\mathcal{P}$. For new tokens ("children," "drawing," "of"), the attention map $M_t^*$ is used. For common tokens ("castle," "river"), the attention map $M_t$ from the original prompt is used.


Figure 9: Illustration of Adding a new phrase in which the composition around an object, in this case a car, is changed. Figure taken from [20]

**Attention Re-weighting.**: Users may wish to modify the influence of specific tokens on the generated image. For example, consider the prompt P = "a bright blue butterfly", and suppose the goal is to make the butterfly appear more or less vibrant. This can be done by adjusting the attention map corresponding to the token j* using a scaling factor c $\in [-2, 2]$, which increases or decreases its impact. The attention maps for all other tokens remain unchanged. Mathematically, this is represented as:

$$\text{Edit}\left(M_t, M_t^*, t\right)_{i,j} = \begin{cases} c\left(M_t\right)_{i,j}, & \text{if } j = j^* \\ \left(M_t\right)_{i,j}, & \text{otherwise} \end{cases} \tag{3.4}$$

### 3.2.2 DiffEdit

The work *DiffEdit: Diffusion-based semantic image editing with mask guidance* [12] elegantly utilizes DDIM inversion to perform semantic edits. These are edits in which the edited image $I^*$ retains much of the features of the original image $I$. Given an

image $I$, DiffEdit automatically generates a mask highlighting regions of the input image that need to be edited. The equation that describes inversion is (1.23). More important to understand the paper is the Neural ODE formulation of DDIM inversion, as given by (1.25):

$$d\vec{u} = d\tau(t)\bar{\epsilon}_\theta^{(t)}\left(\frac{\vec{u}}{\sqrt{1+\tau^2}}, t\right)$$

The authors of the paper use equation 1.25 as their starting point. They parametrize the timestep $t$ to be between 0 and 1 , so that t = 1 corresponds to T steps of diffusion in the original formulation. They explain that, as proposed by Song et al. (2021), the ODE can be utilized to encode an image $\vec{x}_0$ into a latent variable $\vec{x}_r$ for a timestep $r \leq 1$, using the boundary condition $\vec{u}(0) = \vec{x}_0$ instead of $\vec{u}(t = 1)$. This encoding process is achieved by applying an Euler scheme up to timestep $r$. Throughout the paper, this process is referred to as **DDIM encoding**, with the corresponding function mapping $\vec{x}_0$ to $\vec{x}_r$ denoted as $E_r$, and $r$ referred to as the encoding ratio.

The authors highlight that, with sufficiently small steps in the Euler scheme, decoding $\vec{x}_r$ approximately reconstructs the original image $\vec{x}_0$. This property is particularly significant in the context of image editing as all the information of the input image $\vec{x}_0$ is encapsulated in $\vec{x}_r$ and can be retrieved through DDIM sampling.

Given the above as preamble, let us consider their framework more closely. The goal is to change specific parts of an image according to a text query while keeping the rest of the image unchanged. This is achieved by inferring a mask that identifies the region needing changes and then guiding the denoising process in the diffusion model.

A text-conditioned diffusion model is used to generate noise estimates for the image. This is done twice: once with a reference text (e.g., "horse") and once with the editing text query (e.g., "zebra"). More formally, let us denote the image by $\vec{x}_0$. $Q_{ref}$ = "horse" is the original prompt and editing text $Q$ = "horse" is the editing prompt. Using equation (1.25), we perform DDIM encoding to obtain noise estimates $\bar{\epsilon}_{ref}^{(t)}$ and $\bar{\epsilon}_Q^{(t)}$ respectively. We then calculate the difference between these two noise estimates to identity which regions of the image are affected by the text query:

$$\Delta\epsilon = |\epsilon_{ref} - \epsilon_Q| \tag{3.5}$$

To make the difference map more robust, Gaussian noise is added with a strength of 50%. Extreme values in the noise predictions are removed, and the differences are averaged over multiple noise samples (e.g., n = 10 ). This can be expressed algorithmically as follows:

**Image Editing with Gaussian Noise and Masked Diffusion**

**Input:** Input image $\vec{x}_0$, number of samples $n$, editing text query $Q$, DDIM encoding function $E_r$, timestep $r$, threshold 0.5.

1. For each sample $i = 1$ to $n$:
   (a) Add Gaussian noise to the image.
   (b) Calculate the difference:

$$\Delta\epsilon_i = \left|\epsilon_{\text{ref}}^{(i)} - \epsilon_Q^{(i)}\right|.$$

2. Compute the averaged difference map:

$$\Delta\epsilon_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n}\Delta\epsilon_i.$$

3. Rescale $\Delta\epsilon_{\text{avg}}$ to the range $[0, 1]$.

4. Binarize the difference map using the threshold:

$$M(x, y) = \begin{cases} 1 & \text{if } \Delta\epsilon_{\text{avg}}(x, y) > 0.5, \\ 0 & \text{otherwise.} \end{cases}$$

5. Encode the input image $\vec{x}_0$ using the DDIM encoding function $E_r$ at timestep $r$ without any text conditioning.

6. Decode the latent $\vec{x}_r$ back into an image using the diffusion model, conditioned on the editing text query $Q$.

7. During the denoising process, apply the mask $M$ to ensure that only the masked regions of the image are edited.

### 3.2.3 Masa-Ctrl

**Key-Value Injection in Diffusion Models**

Key-Value (KV) Injection is a technique used in diffusion pipelines where features $f_{ref}l$ from a reference image $I_{ref}$ are injected into the self-attention layers of a target image $I_{target}$. Suppose the Diffusion process is carried out until $t = 1, ...T$ steps. For each step $t$, we have features $f_l^t$ emerging from the layer of U-Net. Let $f^l_{\text{target}}$ be the feature map of the target image and $f^l_{\text{ref}}$ be the feature map of the reference image.

In KV injection, **self-attention** is replaced by the following pipeline: The query from the target image at layer $l$ $Q^l_{\text{target}} = W f^l_{target}$ encodes the spatial structure of the target. The key and value from the reference image at layer l $f^l_{\text{ref}}$ and $V^l_{\text{ref}}$ influence the target's appearance based on the reference. The original self-attention (1.49 ) is modified and now reads:

$$\text{Self-Attention } = \left(Q^l_{\text{target}}, K^l_{\text{ref}}, V^l_{\text{ref}}\right) = \text{softmax}\left(\frac{Q^l_{\text{target}} \left(K^l_{ref}\right)^T}{\sqrt{d}}\right) V^l_{\text{ref}} \quad (3.6)$$

Where,

$$Q^l_{target} = W_Q f^l_{target} \quad K^l_{ref} = W_k f^l_{ref} \quad V^l_{ref} = W_v f^l_{ref}$$

Although we introduce KV-injection in context of **Masa-Ctrl**, it is important to note that the technique is more general in the sense that it has been used to carry out other forms of edits such as style transfer.

In *MasaCtrl: Tuning-Free Mutual Self-Attention Control for Consistent Image Synthesis and Editing* [7], the aim of the authors is to alter poses of the same object without effecting the overall content. In order to carry this out, MasaCtrl transforms self-attention into a new type of *mutual self-attention* which uses source images as references to maintain visual consistency.

The authors utilize DDIM inversion to perform their edit. Let $\{\mathcal{P}, \mathcal{P}'\}$ be the original prompt and the modified prompt describing $\{I_S, I_t\}$ respectively. Note that there is no target image $I_t$ in the direct sense before the process. Instead, we would carry out the Masa-CTR pipeline to obtain our target image $I_t$ in accordance with the prompt $\mathcal{P}'$

Given a source image $I_s$, we first encode it into a latent representation $\vec{z}_s$. To enable editing, the source latent $\vec{z}_s$ undergoes DDIM inversion conditioned on $\mathcal{P}$. The result

is $\vec{z}_S^T-$ a noise-infused latent representation that retains the "identity" of the source image but allows for flexible transformation in subsequent steps.

We now take another latent $\vec{z}_t$ and noise it for T steps to obtain $\vec{z}_t^T$. We would now run the backward process to denoise $\vec{z}_t^T$ to obtain $\vec{z}_t^0$ using the prompt $\mathcal{P}'$. During this denoising process, instead of using keys $K_t$ and values $V_t$ from the target latent $\vec{z}_t^i$, the keys $K_s$ and values $V_S$ of the source latent $\vec{z}_s^i$ are utilized. This follows the key-value injection given by equation 3.6.

In practice, applying mutual self-attention across all layers and denoising steps would lead to the target image replicating the source image too closely. To control this, Mutual self-attention is applied selectively, usually in the decoder layers of the U-Net (where high-resolution details are formed). It is activated only after a certain denoising step S allowing the target image's layout to be initially guided by the target prompt. This can be described as:

$$\text{EDIT} := \begin{cases} \text{MutualSelfAttention } (Q_t, K_S, V_S) \text{ if } t > S \text{ and } l > L \\ \text{SelfAttention } (Q, K, V) \text{ otherwise} \end{cases} \tag{3.7}$$

**Foreground-Background Confusion in MasaCtrl**

Self-attention layers in diffusion models capture relationships between different regions of an image, but they don't inherently understand semantic boundaries (e.g., distinguishing a cat from a carpet when both are white). If the foreground and background have similar textures or colors, the model's attention maps might overlap or mix these regions, causing features to "bleed" from one area to another. To overcome this, the authors utilize the *cross-attention maps* of diffusion model. When generating an image from text, cross-attention maps link regions of the image to specific tokens in the prompt. For example, in the prompt "a dog on grass," there would be cross-attention maps that highlight regions corresponding to "dog" and "grass.". These maps reflect which parts of the image are associated with each word in the prompt. By averaging the cross-attention maps associated with these tokens across multiple attention heads and layers, we can generate a mask for the foreground (e.g., the dog) and a separate mask for the background (e.g., the grass). With separate masks for the foreground and background, mutual self-attention can selectively apply source content to the appropriate areas.

Let M be the foreground mask and $1 - M$ be the complement of the foreground mask. Then the mutual self-attention outputs for the foreground and background

regions can be computed separately. The Foreground Self-attention assumes the form:

$$f_{\text{foreground}} = \text{Attention}\left(Q, K_S, V_s; M\right) = \text{sofmax}\left(\frac{QK_s^T}{\sqrt{d}}\right) V_S.M \tag{3.8}$$

This output focuses on the foreground area and applies only to regions where M = 1. On the other hand, for the background, we have:

$$f_{\text{background}} = \text{Attention}\left(Q, K_S, V_s; 1-M\right) = \text{sofmax}\left(\frac{QK_s^T}{\sqrt{d}}\right) V_s.(1-M) \tag{3.9}$$

The combined output assumes the following form:

$$f = f_{\text{background}} + f_{\text{foreground}} \tag{3.10}$$

This combined attention output ensures that the model retains clear distinctions between foreground and background areas with each area sourcing only the relevant content from the source image.

## 3.3 Spatial Layout Control

Spatial layout control allows for the precise arrangement of objects in generated images. This ensures that the image aligns with a given textual or positional description. In T2I Diffusion Models like Stable Diffusion, the user can specify a global scene using text (e.g., "a sunny day at the beach"), but the exact placement and layout of objects are unpredictable. Spatial relationships between objects (e.g., "the ball near the Labrador's paw") are often ignored or inaccurately interpreted. Furthermore, Text-based prompts can struggle with the relative positions of objects. Therefore, the need for Spatial Layout control using editing schemes.

### 3.3.1 Spa-Text

While prompts such as "dog" or "tree" can explain the overall schematics of image, they fail to describe nuanced object characteristics like "a Labrador wearing a red collar" or "a small blue ball." To mitigate this, besides using a global text description as used in normal T2I pipeline, SpaText [4] introduces **local spatio-textual inputs**. This allows users to specify the precise details of certain objects and areas.

The **spatio-textual matrix** $RST$ is a structured input provided by the user to specify spatial and textual details for image generation. Each entry $RST[i, j]$ in the

matrix either specifies the textual description of the content at pixel $[i, j]$ or is marked as $\emptyset$ (null) if no specific description is provided for that pixel. For instance, if the global description $t_{\text{global}}$ is *"a sunny day at the beach,"* the $RST$ matrix could specify Region A containing "a brown dog sitting on the sand.", Region B containing "a red umbrella." and other pixels as $\emptyset$.

The spatio-textual matrix $RST$ is created differently during the training and inference stages. The overall goal however in both cases is to align semantic embeddings with their spatial regions in the image. During the training stage, $RST$ is generated using a panoptic segmentation model to divide the input image $x$ into $N$ segments $\{S_1, S_2, \ldots, S_N\}$. Each segment $S_i$ corresponds to a distinct region or object in the image with a binary mask $M_i$ indicating the pixels belonging to $S_i$. To focus on meaningful regions, small segments that cover less than 5% of the image are excluded. From the remaining segments, $K$ segments are randomly selected for further processing to introduce variability and improve generalization during training.

For each selected segment $S_i$, the model crops a tight bounding box around the segment, masking out all other pixels to ensure that the embedding captures only the content of $S_i$. This cropped region is resized to match the input size required by the CLIP image encoder ($\text{CLIP}_{\text{img}}$). For example, in the figure 12, the image of a dog chasing a ball, the selected segments are $s_1 = \text{dog}$ and $s_2 = \text{ball}$ and these are resized to the spatial dimension of $CLIP_{img}$

The CLIP encoder generates a semantic embedding $\text{CLIP}_{\text{img}}(S_i)$ for the segment. The embeddings of these segments are then spatially mapped back into the spatio-textual matrix $ST_x$ according to their original positions in the image. For every pixel $(j, k)$ in the image, if it belongs to the segment $S_i$, the corresponding entry in $ST_x$ is set to $\text{CLIP}_{\text{img}}(S_i)$. Pixels outside the selected segments are filled with a zero vector $\vec{0}$. In the figure 12), this can be seen with the embedding of $s_1$ populating all the regions in the RT matrix where $s_1$ resides. That is the case for $s_2$ too. We can express this as:

$$ST_x[j, k] = \begin{cases} \text{CLIP}_{\text{img}}(S_i), & \text{if } (j, k) \in S_i, \\ \vec{0}, & \text{otherwise.} \end{cases} \tag{3.11}$$

This training stage helps RST understand how specific regions of the image relate to their semantic descriptions. For example, if a segment is labeled as "a brown dog sitting on grass," the training process helps the model associate the embedding for "brown dog" with the corresponding spatial region and its visual features.

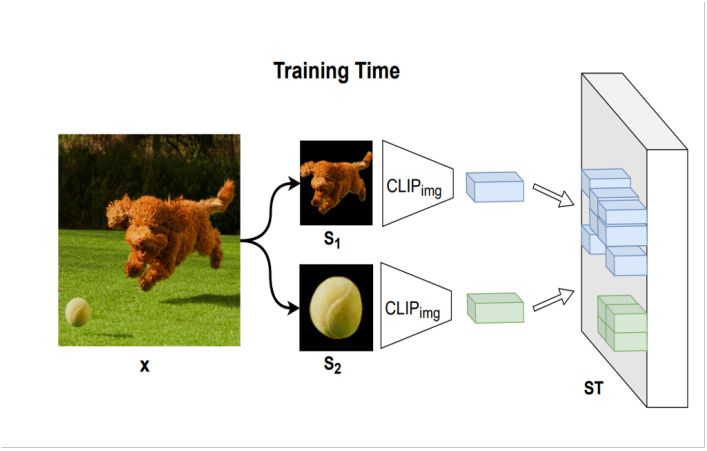


Figure 10: Training Phase of Spatext, figure taken from [4]

During inference, $RST$ is created from user-provided inputs. The user supplies a global text prompt $t_{\text{global}}$, a sparse segmentation map specifying the regions of interest, and free-form textual descriptions for each region. For each region, the textual description $t_{\text{local}}$ is embedded using the CLIP text encoder ($\text{CLIP}_{\text{txt}}$). To ensure compatibility with the embeddings used during training, the text embeddings are transformed into the image embedding space using a prior model $P$. This transformation can be expressed as:

$$P(\text{CLIP}_{\text{txt}}(t_{\text{local}})) \rightarrow \text{CLIP}_{\text{img}}. \tag{3.12}$$

The transformed embeddings are then mapped back into $RST$ using the spatial information from the user-provided segmentation map. Each pixel in a specified region is assigned the embedding corresponding to that region's description. This process creates a sparse spatio-textual matrix where only the described regions are explicitly represented and leave the remaining areas to be inferred by the model.

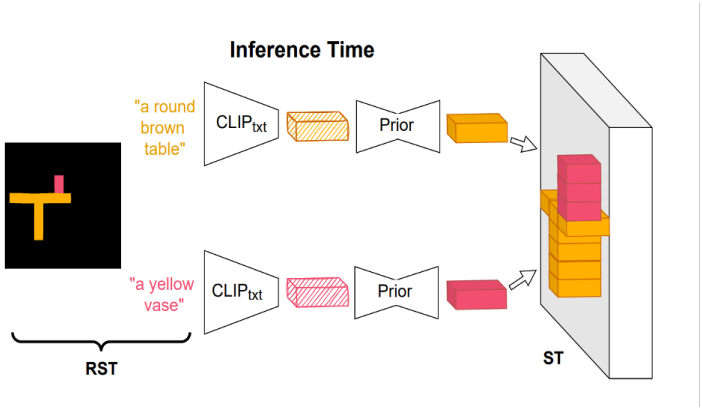


Figure 11: Inference Phase of Spatext, figure taken from [4]

In context of Stable Diffusion, the spatio-textual matrix ($RST$) is incorporated into the generative process by conditioning the latent denoising steps with both global textual prompts and the localized information encoded in $RST$. The dimensions of $RST$ and $x_0$ - the image, are the same. Both of these are downsampled into the latent space of Stable Diffusion by using VQ-VAEs encoder.

At each diffusion step $t$, the noisy latent representation $z_t$ is concatenated with the downsampled $RST$ along the channel dimension. If $z_t$ has a channel depth $C$, and $RST$ has a channel depth $d_{\text{CLIP}}$, the combined input has a shape of:

$$(H, W, C + d_{\text{CLIP}}),$$

where $H$ and $W$ are the spatial dimensions of the latent space and $C$) is the channel dimension of latent space.

The latent denoising U-NeT $f$ takes the concatenated input and produces the next denoised latent $z_{t-1}$ based on the noise schedule $t$, the global text prompt $t_{\text{global}}$, and $RST$:

$$z_{t-1} = f(z_t, \text{CLIP}_{\text{txt}}(t_{\text{global}}), RST, t).$$

The model predicts the noise to be subtracted, ensuring that the output latent aligns with both the global context and the localized details specified by $RST$.

### 3.3.2 LLM-Diffusion

The work *LLM-grounded Diffusion* [27] equips diffusion models with a Large-Language Model (LLM). The LLMs generate scene layouts from user prompts which serve as explicit guides for the image generation process. By generating these structured layouts, the LLM effectively transforms textual descriptions into a **spatial blueprint** that is easier for diffusion models to follow.

Firstly, the LLM are trained using *incontext-learning* which is basically a form of prompt-engineering. In this regime, the model is given a prompt that includes a task description, along with a few examples which serve as a guide for the model to understand the task. The input text prompt from the user $\mathcal{P}$ is converted into an incontext-learning template and then queried to the LLM.

For example, suppose the user gives the prompt: $\mathcal{P}$ = "A realistic photo of a gray cat and an orange dog on the grass". This is embedded into the following in context template for LLM: *"Your task is to generate the bounding boxes for the objects mentioned in the caption, along with a background prompt describing the scene..."* together with In-context examples. An LLM processes this to give an output that may

appear as following:

**LLM Response**

**Caption:** *A realistic photo of a gray cat and an orange dog on the grass.*
**Objects:**

- `'a gray cat'`: [50, 120, 180, 200]
- `'an orange dog'`: [300, 120, 180, 200]
- `'grass'`: [0, 340, 512, 172]

**Background prompt:** *A realistic photo of a grassy outdoor scene.*
**Negative prompt:** *None*

As you can see, the LLM response comprises of two components

1. A captioned bounding box for each foreground object, with coordinates specified in the ( x, y, width, height) format.
2. A simple and concise caption describing the image background along with an optional negative prompt indicating what should not appear in a generated image. The negative prompt is an empty string when the layout does not impose restrictions on what should not appear.

The resulting layout from the LLM completion is then parsed and used for the subsequent image generation process. For each foreground object $i$ in the image layout, the authors first generate an image with a single instance by denoising from $\vec{z}_T^{(i)}$ to $\vec{z}_0^{(i)}$ where $\vec{z}_T^{(i)}$ refers to the latents of object i at denoising timestep $t$. In this denoising process, the format that is followed by the authors for the text prompt is: "[background prompt] with [box caption]" (e.g., "a realistic image of an indoor scene with a gray cat"). The initial noise latent is shared for all boxes to ensure globally coherent viewpoint.

To align the generated object with its bounding box, adjustments are made to the cross-attention maps $A^{(i)}$. Given the cross-attention map $A^i_{uv} = \text{softmax}\left(\vec{q}_u^T \vec{k}_v\right)$, the authors strengthen the cross-attention for pixels inside the box to tokens associated with the box caption. On the other hand, they attenuate the cross-attention from pixels outside the box. This follows previous works which introduce energy-functions to improve generation. In this case, the authors define a simple energy function:

$$E\left(A^{(i)}, i, v\right) = -\operatorname{Topk}_u\left(A_{uv} \cdot \vec{b}^{(i)}\right) + \omega \operatorname{Topk}_u\left(A_{uv} \cdot \left(1 - \vec{b}^{(i)}\right)\right) \tag{3.13}$$

Where the first term tends to increase the value of cross-attention inside the boundind box and the latter term reduces it. The energy function is minimized by updating the latent before each denoising step:

$$\vec{z}_t^{(i)} \leftarrow \vec{z}_t^{(i)} - \eta \nabla_{\vec{z}_t^{(i)}} \sum_{v \in V_i} E\left(A^{(i)}, i, v\right) \tag{3.14}$$

$$\vec{z}_t^{(i)} \leftarrow \text{Denoise}\left(\vec{z}_t^{(i)}\right) \tag{3.15}$$

Where $\eta$ is the guidance strength and denoise $(\cdot)$ denotes one denoising step in the latent diffusion framework.

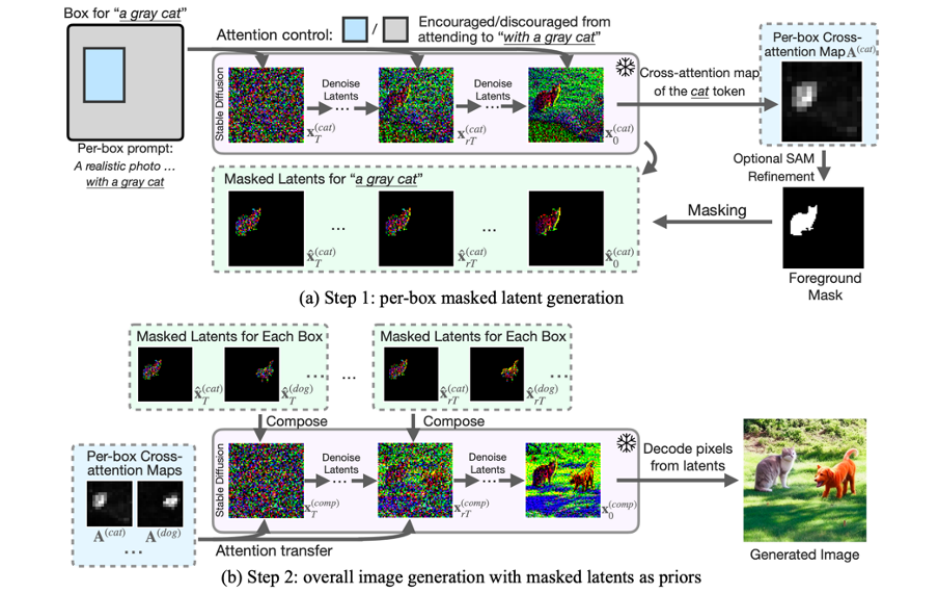


Figure 12: The entire framework of LLM Diffusion, figure taken from [27]

After generation, a cross-attention map that corresponds to the box caption is obtained. This serves as a saliency mask $\vec{m}^{(i)}$ for the object which is optionally refined using SAM or by simple thresholding.

The mask $\vec{m}^{(i)}$ isolates a single object in the scene. This mask is applied to the latent representation at each denoising step through element-wise multiplication which creates a series of masked latents, $\left\{\hat{z}_t^{(i)}\right\}_{t=0}^{T}$:

$$\hat{z}_t^{(i)} = \vec{z}_t^{(i)} \otimes \vec{m}^{(i)}$$

These masked latents act as precise instructions for the diffusion model, essentially saying, "This is where this particular object should go." As the denoising progresses, the model integrates these masked latents into a composite latent representation, $\vec{z}_t^{\text{comp}}$,

which combines all the objects and the background into a single coherent structure:

$$\vec{z}_t^{\text{comp}} \leftarrow \text{LatentCompose}\left(\vec{z}_t^{\text{comp}}, \hat{z}_t^{(i)}, \vec{m}^{(i)}\right) \quad \forall i$$

At the start, $\vec{z}_t^{\text{comp}}$ is initialized with $\vec{z}_T$, ensuring consistency across the entire scene. The function LatentCompose is responsible for placing each masked latent $\hat{z}_t^{(i)}$ in its corresponding spatial location within $\vec{z}_t^{\text{comp}}$, guided by the mask $\vec{m}^{(i)}$.

Diffusion models typically determine object placement early in the denoising process while later steps refine details like textures and shading. To align with this behavior, the latent composition focuses on timesteps between $T$ and $rT^3$, where $r \in [0, 1]$. This ensures that the model prioritizes correct placement of objects without over-constraining the final details/

To further improve alignment, cross-attention maps generated during per-object (per-box) processing are transferred to the composite latent representation. This is done using an enhanced energy function:

$$E^{\text{comp}}\left(A^{(\text{comp})}, A^{(i)}, i, v\right) = E\left(A^{(\text{comp})}, i, v\right) + \lambda \sum_{u \in V_i'} \left|A_{uv}^{\text{comp}} - A_{uv}^{i}\right|$$

Where $\lambda = 2.0$ controls the weighting of the cross-attention alignment, $V_i'$ refers to the token indices associated with the object's description in the text prompt and the energy function ensures that attention for each object aligns with its intended region in the composite latent.

This approach allows the diffusion model to maintain clear boundaries for each object while ensuring the overall scene remains harmonious. Finally, once the composite latent $\vec{z}_0^{\text{comp}}$ is fully constructed, it is decoded into pixel space using the model's image decoder as done in normal Diffusion Pipeline

### 3.3.3 Dense T2I Generation

The work "Dense Text-to-Image Generation with Attention Modulation" [25] provides users with controllability for prompts which *contain a large number of concepts*. Thus, the work is specifically catered to deal with *dense captions* and how to provide better control over each of the sub-concept present in such a caption. To begin with, the work breaks down the dense prompt $\mathcal{P}$ into a set of non-overlapping segments. For example, consider the prompt $\mathcal{P}$ = "A painting of a couple holding a yellow umbrella ($\vec{c}_1$) in a street on a rainy night. The woman is wearing a white dress ($\vec{c}_2$) and the man is wearing a blue suit ($\vec{c}_3$).

It then aims to map each non-overlapping caption $\vec{c}_i$ with an associated binary map

$\vec{m}_i$ where 1 in the binary map indicates the presence of a feature in a specific area of the image and 0 indicates that the feature is not present in the image. The schematic of this for our caption is shown in the figure below:

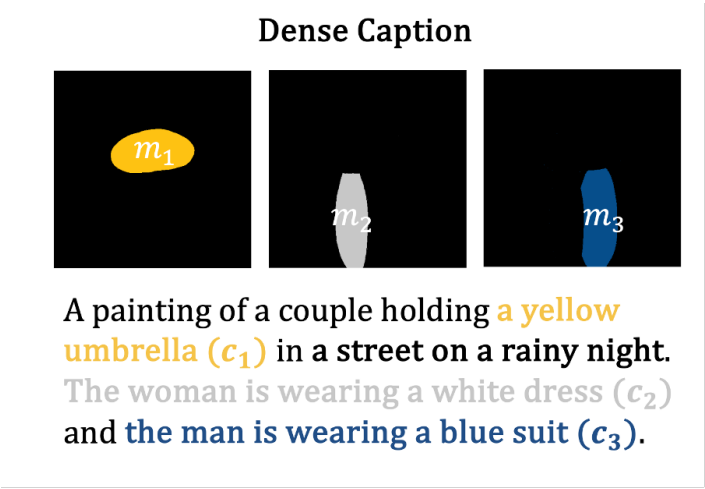


Figure 13: Understanding how segments are associated with each caption, work by [25]

Formally, the condition is defined as a set of N segments $\{(\vec{c}_n, \vec{m}_n)\}_{n=1}^{N}$ where each segment:

$$(\vec{c}_n, \vec{m}_n) \tag{3.16}$$

In equation (3.16), it is important to note that the dimensionality of $\vec{m}_n$ will be determined by patches. For example, suppose we have an image $\mathcal{J} \in \mathbb{R}^{HxWx3}$ and we patchify this into $(P, P)$ . Then $\vec{m}_n$ would also be a 2D matrix of size $(P, P)$. For example, if an image is converted into $(16, 16)$ patches, then a binary $\vec{m}_n$ associated with an object placed on the left of the image might have 1 's there and 0 everywhere else.

Given the input conditions, the aim is to modulate attention maps of all attention layers so that the object described by $\vec{c}_n$ can be generated in the corresponding region $\vec{m}_n$. In order to modulate the attention for N segments consisting of $(\vec{c}_n, \vec{m}_n)$, the authors introduce a **condition map** $\boldsymbol{R}$. The condition map $\boldsymbol{R}$ defines whether to increase or decrease the attention score a particular pair. If two tokens belong to the same segment, they form a positive pair, and their attention score will be increased. If not, they form a negative pair, with their attention decreased. Formally, the matrix is defined as:

$$R_{:j}^{\text{cross}} = \begin{cases} 0 \text{ if } \vec{k}[j] = 0 \\ \vec{m}_{\vec{k}[j]} \text{ otherwise} \end{cases} \tag{3.17}$$

**Defining the Modulation matrix $M$**

With $R^{\text{cross}}$ defined, we can selectively associate a text token with a specific region of interest in the image. However, we would like more control over the values of R (as of now, the value of $R^{\text{cross}}$ is 1 for the region of interest which it describes and zero elsewhere). Towards this aim, the work defines $M_{pos}$ which enhances the attention scores between query and key tokens that belong to the same segment. For example, if want to enhance the region associated with same segment by 0.7 , we would set $M_{pos}[i, j] = 1$ and then take a simple element-wise product of $M_{pos}$ with $R^{cross}$. Overall, we have:

$$\mathrm{M} = R \odot M_{pos}$$

We can incorporate even more control. Suppose we want to reduce the attention values for tokens not in the same segment for a particular text token. To carry this out, we define the following:

$$M = R \odot M_{pos} - (\mathbb{I} - R) \odot M_{neg}$$

Where $M_{neg}$ defines by what amount would we want the regions outside the segment to be attenuated.
The authors further incorporate a matrix S which takes into account the area that each segment occupies in the image. Larger segments (e.g., a large car in the image) would result in higher values in the corresponding areas of S , while smaller segments (e.g., a small umbrella) would result in lower values. Incorporating these, we have:

$$M = R \odot M_{pos} \odot (1 - S) - (\mathbb{I} - R) \odot M_{neg} \odot (1 - S)$$

Finally, the authors observe that a large modification may deteriorate the image quality as the timestep $t$ approaches zero. Therefore, they use a scalar $\lambda_t$ to adjust the degree of modulation using the power function as below:

$$\lambda_t = wt^p \tag{3.18}$$

Incorporating all these effects, the final Modulation matrix M reads:

$$M = \lambda_t \cdot R \odot M_{pos} \odot (1 - S) - \lambda_t \cdot (1 - R) \odot M_{neg} \odot (1 - S) \tag{3.19}$$

The normal cross-attention as computed in U-NeT is given by (1.47). With the

introduction of the modulation matrix, the authors of the paper modify the cross-attention with the modulated attention map:

$$A_{CA} = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d}}\right) \tag{3.20}$$

As for self-attention, which captures how the $i^{th}$ patch correlates with the $j^{th}$, the authors propose a modulation designed to restrict communication between tokens of different segments. This prevents the mixing of features of distinct objects. Specifically, they increase the attention scores for tokens in the same segment and decrease it for those in different segments. The query-key pair condition map $R_{\text{self}}$ for this objective is defined as:

$$R_{ij}^{\text{self}} = \begin{cases} 1 \text{ if } \exists \text{ns.t } \overrightarrow{\text{m}}_{\text{n}}[\text{i}] = 1 \text{ and } \overrightarrow{\text{m}}_{\text{n}}[\text{j}] = 1 \\ 0, \text{ otherwise} \end{cases} \tag{3.21}$$

Which effectively blocks communication with pixels not belonging to the same segment through self-attention.

## 3.4 Text-guided Modification

Text-guided modifications involves giving T21 Diffusion models the capability to change specific parts of an image or the attributes of objects within the image based on natural language descriptions. This might involve changing the *style* while retaining the *structure* of the image, or it might involve changing a specific structure while keeping the overall semantic and style intact.

### 3.4.1 Plug and Play

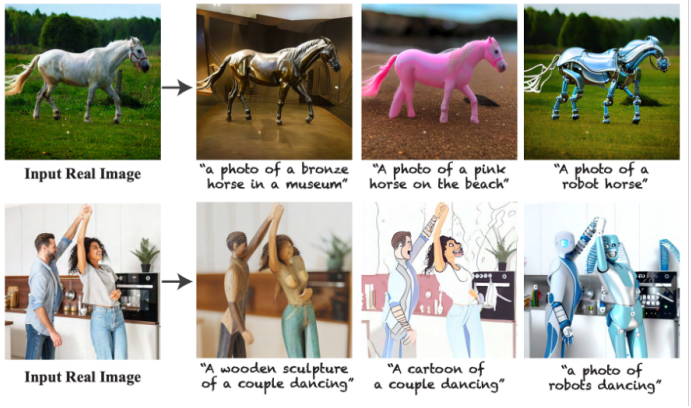


Figure 14: Successful Illustration of Plug & Play Methodology, taken from [36]

The work, *Plug-and-Play Diffusion Features for Text-Driven Image-to-Image Translation* [36] addresses the task of transforming an image into another image based on a textual description. For example, transforming a photo of a house into a version that looks like it was drawn by a child based on the prompt "children's drawing of a house.". Formally, given an input guidance image $I^G$ and a target prompt $\mathcal{P}$, the goal of the paper is to generate a new image $I^*$ that complies with $\mathcal{P}$ and preserves the structure and semantic layout of $I^G$.

To begin with, consider the guidance image $I^G$. We pass it through the VQ-VAE's encoder to obtain the latent $\vec{z}_0$. Instead of noising it to obtain the noisy latent $\vec{z}_T$ as we do in Img-to-Img, we perform DDIM inversion (1.26) in which a U-NeT is used to estimate the noisy latent $\vec{z}_T$. Thus, we obtain:

$$\vec{z}_T^G = \text{Inversion}\left(I^G\right) \tag{3.22}$$

Where $\vec{z}_T^G$ is the noisy latent obtained by inverting the guidance image $I^G$. As explained, DDIM inversion ensures that the latent vector $\vec{z}_T^G$ retains much of the features of the original image in comparison to normal forward diffusion process which destroys the structure of the laten. This latent $\vec{z}_T^G$ noise serves as the starting point for generating a new image $I^*$ based on a different prompt $\mathcal{P}$.

For all time-steps $t = T, ...0$, we use the noisy latent to obtain the estimates:

$$\vec{z}_{t-1}^G = \epsilon_\theta(\vec{z}_t^G, \emptyset, t) \tag{3.23}$$

This process extracts the guidance features $\left\{f_t^l\right\}$ where $l$ denotes the layer of the model with $l = 1$ being the downsampling ResNet Block and $l = 7$ being the upsampling ResNet block. The features capture the spatial and semantic information of the guidance image as it is being refined from noise.

Now, in a separate denoising process, the model generates a new set of features $\left\{f_t^{l*}\right\}$ based on the noisy image $z_t^*$. In the Plug and Play formalism, in order to transfer the features from guidance image $I^G$ onto this generation, the features $\left\{f_t^l\right\}$ from the guidance image are injected into the denoising steps of $z_t^*$. This overrides the features $\left\{f_t^{l*}\right\}$ which were being produced during the generation of $z_t^*$. This operation can be expressed as:

$$\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, \tau^\theta(\vec{c}), t; \left\{f_t^l\right\}\right) \tag{3.24}$$

On the other hand, in case of no injection, we have:

$$\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, P, t; \emptyset\right) = \varepsilon_\theta\left(\vec{x}_t^*, \tau^\theta(\vec{c}), t\right) \tag{3.25}$$

In the paper, the authors utilize Figure 5(a) to demonstrate the impact of injecting features $\{f_t^l\}$ on the overall image generation. They note that injecting features solely at layer $l = 4$ is inadequate for maintaining the structural integrity of the guidance image. As features are injected into progressively deeper layers, the structural preservation improves; however, this also results in the appearance information bleeding into the generated image (e.g., the red t-shirt and blue jeans shades become visible in Layers 4-11). To strike a better balance between preserving the structure of $I^G$ and minimizing the carryover of its appearance, the authors avoid modifying spatial features in the deeper layers and instead rely on the self-attention layers.

Hence, the authors consider the attention matrices $A_t^{l^*}$ to achieve fine-grained control over the generated control. On carrying out the leading principal components $A_t^{l^*}$ for a given image, they observe that in early layers, the attention is aligned with the semantic layout of the image. Gradually, higher-frequency information is captured. Practically, injecting the self-attention matrix is done by replacing the matrix $A_t^{l^*}$ by the modified attention matrix $A_t^l$. Intuitively, this operation pulls features close together, according to the affinities encoded in $A_t^l$. The process is denoted as:

$$\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, P, t; \quad f_t^4, \left\{A_t^l\right\}\right) \tag{3.26}$$

In figure 5(c), the authors show the effect of attention-injection: , $\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, P, t, \left\{A_t^l\right\}\right)$. It is observed there is no semantic association between the original content and the translated one which results in large deviations in structure. Thus, the incorporation of both **feature injection** and **attention injection** is crucial in a proper image translation.

The plug-and-play diffusion features framework is summarized in Alg. 1 below, and is controlled by two parameters: (i) $\tau_f$ defines the sampling step t until which $f_t^4$ are injected. (ii) $\tau_A$ is the sampling step until which $A_t^l$ are injected:

## 3.5 Attribute Level Editing

Attribute-Level Editing in diffusion models reflects how humans manipulate visual information. These work focuses on fine-tuning specific characteristics of objects such as adjusting facial expressions, changing hair color, or modifying clothing styles. This aligns with feature-based attention where humans selectively focus on specific attributes—like the shape of a smile or the color of hair—while ignoring irrelevant details. Many of these works in these regards also confront problem of *attribute binding* and *catastrophic neglect* - problems we discussed before.

**Algorithm 4** Plug-and-Play Diffusion Features

**Inputs:** $I^G$: real guidance image, $P$: target text prompt, $\tau_f, \tau_A$: injection thresholds
**Initialization:**

- $\boldsymbol{x}_T^G \leftarrow$ DDIM-inv$(I^G)$ ▷ Invert guidance image into latent space
- $\boldsymbol{x}_T^* \leftarrow \boldsymbol{x}_T^G$ ▷ Start from the same seed

**for** $t = T, T-1, \ldots, 1$ **do**
  $\boldsymbol{z}_{t-1}^G, \boldsymbol{f}_t^4, \{\boldsymbol{A}_t^l\} \leftarrow \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t^G, \varnothing, t)$
  $\boldsymbol{x}_{t-1}^G \leftarrow$ DDIM-samp$(\boldsymbol{x}_t^G, \boldsymbol{z}_{t-1}^G)$
  **if** $t > \tau_f$ **then**
    $\boldsymbol{f}_t^{*4} \leftarrow \boldsymbol{f}_t^4$
  **else**
    $\boldsymbol{f}_t^{*4} \leftarrow \varnothing$
  **end if**
  **if** $t > \tau_A$ **then**
    $\boldsymbol{A}_t^{*l} \leftarrow \boldsymbol{A}_t^l$
  **else**
    $\boldsymbol{A}_t^{*l} \leftarrow \varnothing$
  **end if**
  $\boldsymbol{z}_{t-1}^* \leftarrow \hat{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}_t^*, P, t; \boldsymbol{f}_t^{*4}, \{\boldsymbol{A}_t^{*l}\})$
  $\boldsymbol{x}_{t-1}^* \leftarrow$ DDIM-samp$(\boldsymbol{x}_t^*, \boldsymbol{z}_{t-1}^*)$
**end for**
  **Output:** $I^* \leftarrow \boldsymbol{x}_0^*$

### 3.5.1 Attend and Excite

*Attend and Excite* [9] addresses the problem of *"catastrophic neglect"* and *"attribute binding"*. To help mitigate these problems, the authors introduce an attention-based formulation dubbed Attend-and-Excite.This guides the model to refine the cross-attention units to attend to all subject tokens in the text prompt and excite their activations.

Let N be the number of text tokens in the prompt. If $(P \times P)$ denotes the number of patches in cross-attention, then $P \in \{64, 32, 16, 8\}$ across the layer of Stable Diffusion II. An attention map $A_t \in \mathbb{R}^{P \times P \times N}$ is calculated over linear projections of the intermediate features $(Q)$ and text embedding $(K)$. $A_t$ defines a distribution over the text tokens for each spatial patch $(i, j)$. Specifically, $A_t[i, j, n]$ denotes the probability assigned to token n for the $(i, j)^{th}$ spatial patch of the intermediate feature map. Intuitively, this probability indicates the amount of information that will be passed from token $n$ to patch $(i, j)$. The authors operate over ( $16 \times 16$ ) attention units since they have been shown to contain the most semantic information.

Intuitively, for a subject to be present in the synthesized image, it should have a high influence on some patch in the image. As such, the authors define a loss objective that attempts to maximize the attention values for each subject token. They then update the noised latent at time $t$ according to the gradient of the computed loss. This encourages the latent at the next timestep to better incorporate all subject tokens in its representation. This manipulation occurs on the fly during inference (i.e., no additional training is performed)

Let $\mathcal{P}$ be a text prompt that guides the image generation process and $\mathcal{S}$ be the Subject token indices which are extracted from the text prompt. These indices $\mathcal{S}$ identify the specific parts of the text that should be emphasized in the image generation process. Use the stable diffusion model SD to compute the attention map $A_t$ at the current timestep t given prompt $\mathcal{P}$ and $\vec{z}_t$.

Now, the authors note that Stable Diffusion learns to consistently assign a high attention value to the $\langle sot \rangle$ (start of text) token in the token distribution defined in $A_t$. Since we are interested in enhancing the actual prompt tokens, they re-weigh the attention values by ignoring the attention of $\langle sot \rangle$ and performing a Softmax operation on the remaining tokens. After the Softmax operation, the $(i, j)$-th entry of the resulting matrix $A_t$ indicates the probability of each of the textual tokens being present in the corresponding image patch. For each subject, they calculate the corresponding attention values $A_t^S$.

**Rationale for applying Gaussian Filter over Attention Maps**

The authors note that the attention maps $A_t^S$ may not fully reflect whether an object is generated in the resulting image.Specifically, a single patch with a high attention value could stem from partial information being passed from the token $s$. This may occur when the model does not generate the full subject, but rather a patch that resembles some part of the subject, e.g., a silhouette that resembles an animal's body part.

To avoid such solutions, the authors apply a Gaussian filter over $A_t^s$. After doing so, the attention value of the maximally activated patch is dependent on its neighboring patches since each patch becomes linear combination of its neighboring patches in the original map. This "distributes" the attention weights and ensures a more robust representation.

Intuitively, successfully generated subjects should have an image patch that significantly attends to their corresponding token. For each subject token in $S$, the optimization loss introduces encourages the existence of at least one patch of $A_t^S$ with a high activation value. Therefore, they define the loss quantifying this desired behavior as:

$$\mathcal{L} = \max_{s \in S} \mathcal{L}_s \text{ where } \mathcal{L}_s = 1 - \max\left(A_t^S\right) \tag{3.27}$$

**Iterative Latent Refinement**. A single latent update has been made at each denoising timestep so far. However, if the attention values of a token do not reach a certain threshold in the early denoising stages, the corresponding object will not be generated. To address this issue, $z_t$ is iteratively updated until a predefined minimum attention value is achieved for all subject tokens. However, performing too many updates of $z_t$ may cause the latent to become out-of-distribution, resulting in incoherent images. Therefore, this refinement is performed gradually across a small subset of timesteps.

Specifically, each subject token is required to reach a maximum attention value of at least 0.8. To achieve this gradually, iterative updates are performed at various denoising steps. The iterations are set at $t_1 = 0$, $t_2 = 10$, and $t_3 = 20$ with minimum required attention values of $T_1 = 0.05$, $T_2 = 0.5$, and $T_3 = 0.8$. This gradual refinement helps prevent $z_t$ from becoming out-of-distribution while encouraging more faithful generations.

### 3.5.2 Structure Diffusion:

The work *"Training-Free Structured Diffusion Guidance for Compositional Text-To-Image Synthesis"* [14] confronts the problem of *attribute binding* and *constitutionality*. For this, they utilize the concept of a **parsing tree**.

**Parsing Trees**

Parsing trees help identify relationships between different parts of a sentence such as which adjectives (attributes) describe which nouns (objects). For example, in the sentence "a red apple," "red" is an attribute and "apple" is the object.

The authors of the paper utilize cross-attention-maps to build what they call *"structured cross-attention guidance"* which utilizes parsing trees. This revealing the hierarchical structure of the language. For example, in the sentence "The cat on the mat is sleeping," the parser would identify "The cat" as a noun phrase, "on the mat" as a prepositional phrase, and "is sleeping" as a verb phrase.

Consider the prompt $\mathcal{P}$: **"A blue chair and a red table in a green room."**. It involves three distinct objects and attributes: A **blue chair**, A **red table** and A **green room**. In the first stage of structure diffusion, **Parsing Trees** are employed to breaks down the sentence into hierarchical noun phrases (NPs): NP1: "blue chair", "red table", NP3: "green room". These phrases represent the key concepts $C = \{c_1, c_2, c_3\}$ in the prompt. Next, a scene graph explicitly represents the relationships between objects.

This structured representation provides a clear mapping of objects, attributes, and their relationships. The structured guidance modifies the cross-attention mechanism during the image generation process to ensure all concepts are faithfully represented.

The attention map $M_t$ is computed for the full prompt $\mathcal{P}$ using the key $K_p$, which represents the entire prompt's text embedding. Separate value tensors $V = [V_p, V_1, V_2, V_3]$ are generated where $V_p$ represents the overall prompt and $V_1, V_2, V_3$ represents individual concepts ("blue chair," "red table," "green room").

The output $O_t$ at each timestep is a weighted combination of the attention map $M_t$ and the value tensors $V_i$:

$$O_t = \frac{1}{k+1} \sum_{i=0}^{k} M_t V_i, \tag{3.28}$$

where $k$ is the number of concepts. This ensures that each concept $c_i$ (e.g., "blue chair") has a dedicated attention focus and the generated image explicitly includes all objects and attributes.

The method uses a loss function to enforce correct attribute binding and compositionality:

$$L = \sum_{c \in C} \left( 1 - \max_{(i,j)} M_t[c] \right), \tag{3.29}$$

where $M_t[c]$ is the Attention map for concept $c$ (e.g., "blue chair") and $\max_{(i,j)} M_t[c]$ ensures that at least one patch in the image strongly attends to the concept.

For our example, the loss would penalize the model if no region in the image attends strongly to "blue chair." Similarly, the loss is computed for "red table" and "green room.". Overall, the loss ensures all objects (chair, table, room) are included. Furthermore, the attributes (blue, red, green) are correctly assigned.

## 3.6 Appearance Transfer

Let $I_T$ represent the *target image* and $I_R$ represent the *reference image*. The goal of appearance transfer is to synthesize a new image $I'_T$, such that:

- $I'_T$ retains the *structural content* of the target image $I_T$.
- $I'_T$ adopts the *appearance attributes* (e.g., color, texture, lighting) of the reference image $I_R$.

### 3.6.1 Cross-Image Attention

The paper *Cross-Image Attention for Zero-Shot Appearance Transfer* [2] uses KV-injection 3.6 for appearance transfer. The cross-image attention mechanism that the work introduces is designed to handle two separate images: one for structure which we denote as $I_{struct}$ and one for appearance which we denote as $I_{app}$. The approach they utilize transfer appearance features from one image onto another while respecting the layout or pose of the target image.

More tangibly, suppose we have a structure image $I_{struc}$of a giraffe with long legs and a distinct neck structure. We also have a appearance image $I_{app}$ with a zebra with stripped patterns. During cross-image attention mechanism, the queries of $I_{struct}$ (giraffe's structure) will interact with keys and values from $I_{app}$ (zebra's appearance). For instance, a query on the giraffe's neck might attend to the zebra's body, aligning their semantic roles (e.g., elongated neck/striped body) (see figure below)

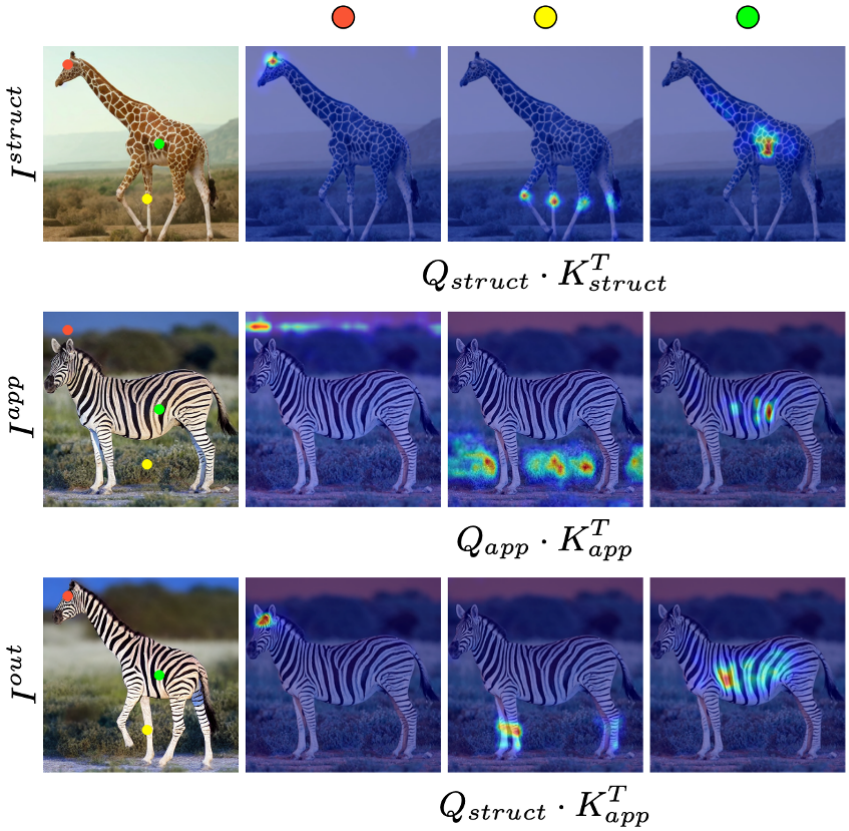


Figure 15: Schematic of cross-image attention mechanism which utilizes KV injection, figure taken from [2]

The attention map is adjusted to enhance specific correspondences (e.g., giraffe neck aligns sharply with zebra body, as the figure 15 highlights). To sharpen the cross-image attention maps and ensure they are focused, the authors introduce a contrast enhancement step. This process increases the variance in attention weights which helps the model attend more distinctly to relevant regions.

Classifier-free guidance is used to further align the appearance transfer with the desired characteristics of the target image. This controls the extent to which appearance features from $I_{app}$ influence the final output. For appearance guidance, they modify the cross-image attention by blending it with unconditioned (structure-only) attention:

$$\text{Attention}_{\text{guided}} = (1+w).\ \text{Attention}_{\text{cross}} - w \cdot \text{Attention}_{\text{uncond}} \tag{3.30}$$

AdaLN is utilized to ensure that the feature maps $F_{\text{struct}}$ are adjusted to match the appearance statistics of appearance features $F_{\text{app}}$. The transformation reads:

$$\text{AdaLN}\left(F_{\text{struct}}, F_{\text{app}}\right) = \sigma\left(F_{\text{app}}\right)\left(\frac{F_{\text{struc}} - \mu\left(F_{\text{struct}}\right)}{\sigma\left(F_{\text{struct}}\right)}\right) + \mu\left(F_{\text{app}}\right) \tag{3.31}$$

### 3.6.2 Eye-for-an-Eye

The paper *Eye-for-an-eye: Appearance Transfer with Semantic Correspondence in Diffusion Models* [17] builds on previous works of using KV-injection for appearance transfer. It introduces a systematic way to transfer the appearance from a reference image to a target image while ensuring that the transfer respects the semantic regions of the target image.

During each timestep of the diffusion process, feature maps are extracted from specific layers of the U-Net for both the target image and reference image. For each pixel $q$ in the target image feature map $F_{\text{target}}$, the method finds a corresponding pixel $p$ in the reference image feature map $F_{\text{ref}}$ by maximizing their cosine similarity. This finds the pixel in $F_{ref}$ that best matches the appearance of q in $F_{\text{target}}$, aligning textures based on content rather that proximity.

$$p = \arg\max_{p \in [0,h]x[0,w]} \text{sim}\left(F_{\text{target}}\left(q\right), F_{\text{ref}}\left(p\right)\right) \tag{3.32}$$

This similarity-based matching creates a semantic alignment between features in the two images, enabling textures or colors to map onto corresponding areas (e.g., a zebra's stripes onto a giraffe's body). Once the correspondences are established, the reference features are rearranged to match the spatial arrangement of the target image, resulting in $\widetilde{F}_{ref}$. This realignment ensures that the transferred features map semantically to the correct regions in the target image.

To integrate the rearranged features into the target image, they use a mask that selectively transfers appearance only to relevant regions of the target. This mask helps separate the foreground and background to prevent background features from contaminating the object features (e.g., applying sky texture onto an animal's body).

$$F^{\text{target}}_{\text{masked}} = F_{\text{target}}\left[M_{\text{target}}\right], F^{\text{ref}}_{\text{masked}} = F_{\text{ref}}\left[M_{\text{ref}}\right] \tag{3.33}$$

Here, only the masked regions are considered for matching, ensuring that only semantically meaningful features are aligned. After obtaining semantically aligned features, the method injects them back into the target image feature map:

$$F'_{\text{out}} = F^{\sim}_{\text{ref}} \odot M_{\text{target}} + F_{\text{out}} \odot \left(1 - M_{\text{target}}\right) \tag{3.34}$$

Where $F_{ref}^{\sim}$ represents the rearranged reference features based on semantic matching. The modified self-attention then uses:

$$\text{Self-attn}\left(F'_{\text{out}}\right) = \text{softmax}\left(\frac{Q'_{\text{out}}\ \left(K'_{\text{out}}\right)^T}{\sqrt{d_k}}\right) V'_{\text{out}} \tag{3.35}$$

Where $Q'_{\text{out}}, K'_{\text{out}}$ and $V'_{\text{out}}$ are derived from $F'_{\text{out}}$. Finally, Adaptive Instance Normalization (AdaIN) is applied just as it is in cross-image attention to balance the brightness and color contrast between the reference and target features:

$$\text{AdaLN}(F) = \sigma_{\text{ref}} \frac{\left(F - \mu_{\text{target}}\right)}{\sigma_{\text{target}}} + \mu_{\text{ref}} \tag{3.36}$$

This standardizes the appearance features of the target with the mean and variance from the reference, smoothing out color and brightness discrepancies.

# 4 Personalization:

## 4.1 Future Works and Ideas

1. Read ProSpect: Prompt Spectrum for Attribute-Aware Personalization of Diffusion Models

2. Read Dreambooth 3. Read the CMU work where they aim to perform customization in one-shot. 4. Read "Multi-Concept Customization of Text-to-Image Diffusion 5. Isn't personalization in some sense capturing the manifold of the embedding? can't we introduce some sort of noise methodology to ensure that the embeddings learned are robust? 6. Make an extensive section on editability vs distortion 7. With **"Break-A-Scene"**, can't we combine unsupervised segmentation methods like **DiffAttend** and correspondence method to improve personalization? For example, suppose that I have a collection of images $I$ consisting of a person. If I am able to perform unsupervised segmentation and correspondense to focus on "eye", I can train a specific embedding to capture "eye". 8. Incorporate Reversion II also. 9. Read "A Neural Space-Time Representation for Text-to-Image Personalization"

## 4.2 GAN-approaches

Personalization as a concept predates the emergence of diffusion models and finds its origins in advancements made with GANs. Early approaches in context of GANs focused on adapting pretrained models to generate content specific subjects or styles.

Among these, Few-Shot GAN Adaptation [26] introduced a framework for fine-tuning GANs on small datasets (e.g., 10-20 images) without overfitting by using Elastic Weight Consolidation (EWC). This technique identified the most critical weights in the pre-trained GAN and penalized large updates to them. This ensured the model generates subject-specific images while retaining its ability to generate diverse outputs. Similarly, Instance-Conditioned GANs extended GANs by conditioning the generation process on instance-level embeddings. This enabled the model to synthesize variations of specific objects or subjects [8]. These methods were effective for generalization across similar instances but often lacked the fine-grained detail required for unique self-defined subjects.

The introduction of StyleGANs [24] significantly advanced the personalization paradigm. It gave rise to latent space manipulation [1] and the subsequent usage of pivotal tuning [30] to further optimize the former methodology and address the weaknesses arising from it. These approaches exploited the disentangled latent space of StyleGAN to allow detailed subject-specific edits and stylistic adjustments. Since both of these methodologies share similarities with the subsequent approaches used to perform personalization in Diffusion Models, a closer examination of both of the approaches will be help.

To begin with, let us give a high overview of StyleGANs. In traditional GANs, the latent vector $\vec{z}$ is sampled from a fixed distribution and directly fed into the generator's input layer. The generator then transforms $\vec{z}$ into an image using a series of fully connected and convolutional layers. This direct mapping from $\vec{z}$ to image $\mathcal{J}$ lead to entangled and poorly separated features in the latent space. StyleGAN introduces an intermediate latent space $\vec{w}$ which is a transformed version of $\vec{z}$. Instead of directly using $\vec{z}$, it passes $\vec{z}$ through a mapping network f to produce $\vec{w}$, which controls the style at different layers of the generator. Therefore, instead of the direct map:

$$G : \vec{z} \rightarrow \mathcal{J}$$

We now have an intermediate map f , usually in the form of MLP:

$$f : \vec{Z} \rightarrow \vec{w}$$
$$G : \vec{w} \rightarrow \mathcal{J}$$

The mapping network helps disentangle features and provides better control over different aspects of the image. For example, $\vec{w}$ might control high-level attributes like pose at some layers and fine details like color at others. This leads to a more interpretable and manipulatable latent space.

The structure of StyleGANs is shown below:

**INSERT FIGURE HERE**

StyleGAN introduces Adaptive Instance Normalization (AdaIN) to gain fine-grained control over the image at multiple levels of abstraction. The style vector $\vec{w}$, after being transformed by the mapping network $f(\vec{z})$, is applied to each layer of the generator through AdaIN, which adjusts the mean and variance of the feature maps:

$$\text{AdaLN}(\vec{x}, \vec{w}) = \sigma(\vec{w})\frac{\vec{x} - \mu(\vec{x})}{\sigma(\vec{x})} + \mu(\vec{w})$$

Where $\vec{x}$ is the feature map and $\vec{w}$ controls the style at each resolution level.
In StyleGAN, random noise is added at each layer of the generator independently from the latent vector $\vec{w}$. This noise is injected into the feature maps to introduce stochastic variations in the generated images, particularly affecting fine details like hair strands or skin texture:

$$\vec{x}' = \vec{x} + N$$

Where N is the noise map that adds variation to the feature map $\vec{x}$. To promote disentanglement, StyleGAN uses style mixing regularization during training. It randomly selects two latent vectors $\vec{w}_1$ and $\vec{w}_2$ and applies them at different layers of the generator. For example, $\vec{w}_1$ might control the style at the lower-resolution layers (coarse features), and $\vec{w}_2$ might control the higherresolution layers (fine details):

$$G\left(\text{AdaLN}\left(\vec{x}_1, \vec{w}_1\right), \text{AdaLN}\left(\vec{x}_2, \vec{w}_2\right)\right) \rightarrow \text{ Mixed Styles}$$

Given the structure of Style GANs, we can understand Latent Space Manipulation as introduced by R Abdal et al in 2019. In StyleGAN, there are multiple latent spaces where an input vector can be embedded and these latent spaces are essential for controlling the style and content of the generated images. Z is the initial latent space, where an input vector $\vec{z}$ is sampled from a simple distribution, typically a Gaussian distribution $\mathcal{N}(0, \mathbb{I})$. This space is 512-dimensional. However, it is extremely entangled.

As we saw, W is the intermediate latent space in StyleGAN. The vector $\vec{w} \in$ W is obtained by passing $\vec{z}$ through a learned, fully connected neural network called the mapping network f as we saw above. However, while this space is better suited for controlling style at a high level, it doesn't give fine-grained control at individual layers of the generator. Thus, we see that while Z is too entangled, W space comes with its own limitations: it applies the same latent vector $\text{w}^{\vec{Z}}$ across all layers of the generator. This means that one latent vector controls every layer, limiting the flexibility to adjust

styles independently at different layers.
W + is an extended latent space introduced to overcome the limitations of embedding directly into W. Instead of using a single latent vector $\vec{w}$ for the entire generator, W+ concatenates 18 different 512-dimensional latent vectors (one for each layer of the StyleGAN generator that receives input via AdaIN):

$$W^+ = \{w_1, w_2, \dots w_{18}\}, w_i \in \mathbb{R}^{512}$$

Where the total number of layers in StyleGANs is 18. Each layer now receives a separate latent vector $\vec{w}_i$ rather than all layers being controlled by the same $\vec{w}$. The latent space, $\boldsymbol{W}+$ allows for more layer-specific control over the generated image, meaning you can independently control coarse features (e.g., pose) in earlier layers and finer features (e.g., texture, color) in later layers.

Given the $W^+$space, the authors solve for a latent code $\vec{w}^*$ such that the generate image $G(\vec{w}^*)$ closely resemble the input image $\mathcal{J}$. This is achieved by minimizing a reconstruction loss:

$$\vec{w}^* = \arg\min_{w \in W} \mathcal{L}_{\text{total}} \left(G(\vec{w}), \mathcal{J}\right)$$

Where $\mathcal{L}_{\text{total}}$ includes pixel-wise loss and perceptual loss. Note that the process does not involve the fine-tuning of the Generator $G$ which remains frozen.

While latent-space manipulation achieves editability due to the disentangled nature of $W^+$, this limits reconstruction fidelity. The authors of the paper [35] encapsulate this tension between reconstructing a real image and retaining the ability to perform semantic edits in what they call the "Distortion vs Editability" trade-off. In W space, the generator has low reconstruction fidelity but high editability. On the other hand for $W^+$, we have high fidelity but low editability. The reason for that is that W's generator's strong priors (a single collection of weight is optimized throughout) ensure edits are meaningful and disentangled, but reconstructions suffer because W lacks fine-grained details. With $W^+$, it is the opposite: the generator can better reconstruct real images, but edits become less predictable due to the weights being disentangled.

In Pivotal Tuning [30], the aim is to resolve this "Distortion vs Editability" trade-off to balance both fidelity and semantic editability. To carry this out, the authors fine-tune the generator $G_\theta$ around the pivot latent code $\vec{w}_p$ :

$$\theta^* = \arg\min_{\theta} \mathcal{L}_{PTI} = \mathcal{L}_{\text{Recon}} \left(G_\theta\left(w_p\right), \mathcal{J}\right) + \lambda_{\text{reg}} \mathcal{L}_{reg}(\theta)$$

By fine-tuning only locally around $w_p$, pivotal tuning preserves the disentangled

structure of $W$ and $W^+$, ensuring semantic directions remain well-defined.

Another notable work in context of personalization is [29]. The methodology enables text-driven personalization of StyleGAN-generated images by aligning the StyleGAN latent space with CLIP's multimodal text-image embedding space. Through optimization or a trained latent mapper, it manipulates latent vectors to match semantic descriptions in text while preserving key attributes like identity or structure. This approach allows users to perform intuitive, fine-grained edits to images based solely on textual guidance.

## 4.3 DreamBooth

Unlike textual inversion which we would study later, DreamBooth [33] fine-tunes **the entire text-to-image diffusion model** embedding the subject directly into the output domain rather than limiting it to the latent embedding space. The pretrained model parameters $\theta$ are fine-tuned to bind a unique identifier $[V]$ with subject-specific features. While the embedding $c$ remains consistent with Textual Inversion, it is now utilized to fine-tune the model itself.

We begin with 3-5 images of the subject captured in varying contexts. These images are labeled with descriptive text prompts such as *"a [V] dog"*, where $[V]$ is a unique identifier that represents the subject. To ensure uniqueness, $[V]$ is chosen as a token rarely present in the model's vocabulary, minimizing interference with existing knowledge.

Let the collection of images $\{\mathcal{J}_1, \mathcal{J}_2, \ldots \mathcal{J}_5\}$ contain the concept that we would like to personalize. The noise latent $\{\vec{z}_1, \vec{z}_2, \ldots \vec{z}_5\}$ are obtained by passing each image to the encoder of VQ-VAE and then performing the forward diffusion process. We then introduce a learnable pseudo-word $[V]$. The model U-NeT must denoise the latents back into a meaningful image conditioned on the text input that contains the pseudo-word $[V]$.

The text-to-image diffusion model (e.g., Stable Diffusion or Imagen) is fine-tuned on the subject's images and their corresponding descriptive prompts. The fine-tuning process uses the diffusion model loss:

$$\mathcal{L}_{\text{noise}} = \mathbb{E}_{x,c,\epsilon,t}\left[w_t \left\|\hat{\epsilon}_\theta(\alpha_t x + \sigma_t \epsilon, c) - \epsilon\right\|_2^2\right] \tag{4.1}$$

where the loss is backpropogated to directly optimize the parameters $\theta$ of the model. This step trains the model to associate $[V]$ with the specific visual characteristics of the subject while preserving its knowledge of what a "dog" generally looks like.

Fine-tuning on a small set of images risks overfitting, where the model might either memorize the exact training images or lose its ability to generate diverse outputs for the subject class (e.g., other "dogs"). To mitigate this, a class-specific prior preservation loss is introduced. This loss supervises the model using both the fine-tuning dataset (specific to the subject) and synthetic samples generated by the pretrained model for the general subject class. The prior preservation loss is:

$$\mathcal{L}_{\text{noise}} = \mathbb{E}_{x,c,\epsilon,\epsilon',t}\left[w_t \left\|\epsilon - \hat{\epsilon}_\theta(\alpha_t x + \sigma_t \epsilon, c)\right\|_2^2 + \lambda w'_t \left\|\epsilon' - \hat{\epsilon}_\theta(\alpha'_t x_{\text{pr}} + \sigma'_t \epsilon', c_{\text{pr}})\right\|_2^2\right]. \quad (4.2)$$

where the first term ensures fidelity to the specific subject while the second term encourages diversity by using the pretrained model's outputs as pseudo-labels. For example, suppose we are personalizing a particular dog. Then, $c_r$ would be "dogs". The model predicts the noise $\epsilon'$ for the generic dog and learns to reconstruct it. This prevents the model from overfitting to the personalized concept and losing the broader understanding of 'dogs'.

After fine-tuning, the model can synthesize the subject in new scenarios using simple text prompts such as *"A [V] dog in the jungle"* or *"A statue of a [V] dog in the style of Michelangelo"*. The model leverages:

- The unique visual features tied to $[V]$ (e.g., the white patch on the dog's forehead).
- Its extensive prior knowledge about jungles, statues, and artistic styles.

This enables the creation of photorealistic and artistic renditions of the subject in diverse contexts.

## 4.4 Textual Inversion for Diffusion Models

### 4.4.1 An Image is worth one word

[15] introduces textual Inversion for Diffusion models. This allows a user to represent a unique concept (e.g., a favorite toy, a personal object, etc.) by finding new pseudo-words in the embedding space of a pre-trained text-to-image model. These pseudo-words are optimized to guide the text-to-image model into generating faithful representations of the unique concept.

In the stable diffusion II pipeline, the text prompt t is tokenized into individual tokens $\{\vec{w}_1, \ldots \vec{w}_n\}$ and each token is embedded using CLIP text encoder. Each token $\vec{w}_i$ maps to a corresponding embedding vector $\vec{e}_i \in \mathbb{R}^d$ in the latent space, where d is

the dimensionality of the embedding. The overall text embedding for the prompt is often denoted as:

$$E(t) = [\vec{e}_1, \dots \vec{e}_n] \in \mathbb{R}^{nxd} \tag{4.3}$$

This embedding sequence conditions the image generation process, influencing attention maps in the U-Net or other architectures.

In Textual Inversion, you introduce new tokens that don't exist in the original vocabulary of the model. These tokens $\vec{w}_{\text{new}}$ represent a new concept or object. Instead of randomly initializing $\vec{e}_{\text{new}}$ for these new tokens, Textual Inversion learns an embedding vector for $\vec{w}_{\text{new}}$ by optimizing it based on a few images of the concept you want to learn.

Given a collection of images $\{\mathcal{J}_1, \mathcal{J}_2, \dots \mathcal{J}_5\}$ containing the concept that we would like to invert, the noise latent $\{\vec{z}_1, \vec{z}_2, \dots \vec{z}_5\}$ are obtained by passing each image to the encoder of VQ-VAE and then performing the forward diffusion process. We then introduce a learnable pseudo-word $\vec{d}$ in the template prompts $\mathbf{D}$. The model U-NeT must denoise the latents back into a meaningful image conditioned on the text input that contains the pseudo-word $\vec{d}$ :

$$\mathcal{L}(\vec{d}) = \mathbb{E}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{z}_t, \vec{d}\right)\right\|_2^2\right]$$

This is a reconstruction loss where given the template prompt $\mathbf{D}$ and a latent $\vec{z}_i$, the model must reconstruct $\mathcal{J}_i$. Initially, the vector is a poor representation of the target concept so the model won't be able to correctly predict the noise associated with the images of that concept, resulting in a higher loss. The optimization adjusts $\vec{d}$ so that the model becomes better at predicting the noise in the latent space when generating images that match the concept represented by $\vec{d}$

The template prompts $\mathbf{D}$ for a *single subject* assume the following form, to state a few of them:

- 'a photo of a } ', 'a rendering of a } ', 'a cropped photo of the } ', 'the photo of a } ',
- 'a photo of a clean } ', 'a photo of a dirty } ', 'a dark photo of the } ', 'a photo of my } ',
- 'a photo of the cool } ', 'a close-up photo of a } ', 'a bright photo of the } '

, Where "}" is the place-holder. On the other hand, the template prompts $\mathbf{D}$ for a *dual subject* assume the following form:

- 'a photo of a } with } ', 'a rendering of a } with } ', 'a cropped photo of the } with }′,
- 'the photo of a } with } ', 'a photo of a clean } with } ', 'a photo of a dirty } with } ',
- 'a dark photo of the } with } ', 'a photo of my } with } ', 'a photo of the cool } with }′,

This optimization process finds an embedding $\vec{d}$ that represents the user-provided concept, and it can then be reused in any text prompt for generating new images. The authors experiments were conducted using 2× V100 GPUs with a batch size of 4 The following method requires 2-3 hours of training on a GPU. Therefore, the entire framework can *easily be carried out by an independent researcher with low-compute resources*. Multiple code implementations are available. For example, see the following Hugging Face tutorial

When it came to experiments, the authors considered four different variations for performing textual inversion:

- **Approach 1:** Instead of using a single embedding vector $\vec{d}^*$ to represent the concept, the authors use multiple vectors to represent the same concept. The idea is to describe the concept through multiple learned pseudo-words, such as using two vectors for a 2-word setup or three vectors for a 3 word setup. The rationale is that the single-vector setup might act as a bottleneck because a single vector could struggle to capture all the fine details of a complex concept. By using multiple vectors, the model can potentially capture a richer representation of the concept, leading to more accurate image reconstructions.
- **Approach 2:** The authors used Progressive extensions. Instead of starting with multiple vectors from the beginning, the authors start with one vector and add more vectors gradually during training. Specifically, they start with a single vector, then add a second vector after 2000 steps, and a third vector after 4000 steps. The idea is to allow the model to first focus on capturing the core details of the concept with one vector. Once those details are learned, additional vectors are introduced to capture finer details of the concept. This staged approach could help the model avoid being overwhelmed by too many vectors from the start and focus on the essential aspects of the concept first.
- **Approach 3:** The authors apply a regularization term that encourages the learned embedding vector $\vec{d}^*$ to stay close to the embeddings of existing words in

the pre-trained model's vocabulary. In practice, this is done by minimizing the L2 distance between the learned embedding and the embedding of a coarse descriptor of the object (e.g., if the object is a sculpture, the coarse descriptor might be the word "sculpture"). This method is inspired by observations in GAN inversion that latent codes which stay closer to the distribution of real-world embeddings (those seen during training) have increased editability. By keeping the learned embedding close to an existing word embedding, the model might become more flexible in editing the concept while preserving the ability to generate realistic images.

- **Approach 4:** In this method, the authors introduce unique tokens for each image in the training set, rather than using a single word embedding for the entire set of images. In addition to a shared placeholder $S_i$ for the concept, they introduce unique placeholders $S_i$ for each image. For example, the concept $S^*$ (the object) is shared across images, while each image i is associated with a unique embedding $S_i$ capturing per-image details like background variations.

### 4.4.2 Extended Textual Inversion

In the work [15], a single embedding **p** is embedded into the U-net. This embedding interacts with the image features $\vec{f_t}$ using cross-attention in the U-NeT across all the layers $l = 1, ...n$. In the work [38] the authors take inspiration from previous works in context of GANs, namely [1] which introduces W+ space to expand the traditional W space. Instead of sending a single embedding **p** into the U-NeT, they send each of the embedding $\{p_1, p_2, \dots p_n\}$ to a particular layer where each layer $p_i \in P$ is specific to the i-th layer of the U-Net. Thus for the prompt p =a red cat and a blue dog, we might have that $p_2$ = "red" is sent to the first layer, $p_3$ = "cat" is sent to the second layer and so on...

Usually, a single embedding vector $p^*$ is optimized in personalization. However with the above paradigm, instead of using a single embedding vector $p^*$ for all layers, we can learn multiple embeddings $\{p_1, p_2, \dots p_n\}$ where each layer $p_i \in P$ is specific to the i -th layer of the U-Net. Each layer learns a different embedding that focuses on certain aspects of the image-coarse layers handle structure, while fine layers handle appearance.

To introduce their methodology, the authors carry out a simple experiment on the publicly available Stable Diffusion model. They partitioned the cross-attention layers of the denoising U-net into two subsets: coarse layers with low spatial resolution

and fine layers with high spatial resolution. They then used two conditioning prompts: "red cube" and "green lizard", and inject one prompt into one subset of cross-attention layers, while injecting the second prompt into the other subset. Notably, at the first run the model generates a red lizard, by taking the subject from the coarse layers' text conditioning, and appearance from the fine layers' conditioning. Similarly, in the second run it generates the green cube, once again taking the appearance from the fine layers and the subject from the coarse layers. This experiment suggests that the conditioning mechanism at different resolutions processes prompts differently, with different attributes exerting greater influence at different levels

Given a set of images $\mathcal{J} = \{I_1, \dots I_k\}$ of a specific subject, the goal of the Textual Inversion (TI) operation is to find a representation of the object in the conditioning space P . In **Extended Textual Inversion (XTI)**, we add n new textual tokens $t_1, \dots, t_n$ to the tokenizer model, associated with n new token embeddings lookup-table elements $e_1, \dots, e_n$. Then, we optimize the token embeddings with the objective to predict the noise of a noisy images from $\mathcal{J}$, while the token embeddings are injected to the network.

Assuming that the denoising U-net is parameterized by a set of parameters denoted by $\theta$, and operates within the extended conditioning space as previously described, the reconstruction objective for the embeddings $e_1, \dots, e_n$. that correspond to the tokens $t_1, \dots, t_n$ as follows:

$$\mathcal{L}_{XTI}(v^*) = \mathbb{E}\left[\|\varepsilon - \varepsilon_\theta(\vec{z}_t, p_1, p_2, \dots p_n)\|_2^2\right] \tag{4.4}$$

Where $\vec{z}_t$ is the image $\mathcal{J}$ noised with the additive noise $\varepsilon$ according to the noise level t . This optimization is applied independently to each cross-attention layer.

To understand the framework, suppose we want to personalize a model to generate images of a customized teapot. For example, **a blue teapot with unique floral patterns** while retaining flexibility to manipulate its **appearance** or **shape**. With $P^+$, the conditioning embeddings are *layer-specific*:

$$P^+ = \{p_1, p_2, \dots, p_n\}$$

where $p_i \in \mathbb{R}^d$ (dimensionality $d$) corresponds to the conditioning embedding for the $i$-th cross-attention layer of the U-Net. The coarse layers $p_1, \dots, p_k$ control shape/structure while Fine layers $p_{k+1}, \dots, p_n$ control *appearance/details*.

In XTI process, we add new textual tokens $t_1, t_2, \dots, t_n$ corresponding to each layer rather a single textual token and we optimize embeddings $e_1, e_2, \dots, e_n$ to reconstruct

the image $I$. We now have a set of layer-specific embeddings $\{e_1, e_2, \ldots, e_n\}$ that encode our blue floral teapot into the $P^+$ space. We can now use these embeddings $\{e_1, e_2, \ldots, e_n\}$ to generate images with layer-wise control. Suppose we want to keep the **shape** of the teapot but change its **appearance** to *golden*. For that, we would inject the shape embeddings $e_1, \ldots, e_k$ (coarse layers) from our personalized teapot and replace the appearance embeddings $e_{k+1}, \ldots, e_n$ (fine layers) with those from a prompt like "golden teapot. The generated image retains the custom shape of the blue floral teapot but now has golden color. On the other hand, we we want to the **appearance** of the floral patterns but modify the **shape** to a cube then we would inject shape embeddings $c_1, \ldots, c_k$ (from "cube") into the coarse layers and keep the appearance embeddings $e_{k+1}, \ldots, e_n$ (from the personalized teapot).

On a broader level, TXI improves the **representation of embeddings** by disentangling the independent variations in the underlying concept c that we would like to embed.

**Using TXI for Style Mixing**

An interesting application of TXI framework is style-mixing. Given two concepts $A$ (e.g., a skull mug) and $B$ (e.g., a golden cat statue), we can generate a new image that combines the **geometry** of concept $A$ and the **appearance** of concept $B$.

We assume that XTI has already been applied to both concepts, resulting in two sets of optimized embeddings:

$$P_A^+ = \{e_{A1}, e_{A2}, \ldots, e_{An}\}, \quad \text{Layer-wise embeddings for concept } A.$$
$$P_B^+ = \{e_{B1}, e_{B2}, \ldots, e_{Bn}\}, \quad \text{Layer-wise embeddings for concept } B.$$

The idea is to inject the shape embeddings and the appearance embeddings $e_{B(k+1)}, e_{B(k+2)}, \ldots, e_{Bn}$ from fine layers of $B$). Up until k layer, we would inject shape embeddings and after that we would inject concept embeddings. This would lead to a mixed embedding $P_{\text{mix}}^+$. When we would condition our T2I diffusion model on this embedding, the resulting image will exhibit shape from concept A and appearance from concept B.

The style mixing process can be generalized to blend multiple concepts. For example, by blending embeddings at different ranges $k$ and $K$, we can control how much of the shape or appearance is borrowed from each concept.

### 4.4.3 Break-A-Scene

The methods discussed so far aim to introduce a user-provided concept to the model which allows its synthesis in diverse contexts. However, they primarily focus on the case

of learning a single concept from multiple images with variations in backgrounds and poses. In [3], the authors introduce effective textual inversion given a **single image** of a scene that may **contain several concepts**. To do so, the authors augment the input image with masks provided by the user or by a pre-trained segmentation model. They then perform two-phase customization processes that optimizes a set of dedicated textual embeddings and the model weights:

1. Token Optimization (Phase 1)

2. Weights Optimization (Phase 2)

Note that the above does not strictly **obey the textual inversion paradigm**. Instead, both the model parameters and token is being optimized so it is a mix of both methodologies - something that would be characteristic now of many of the works that we will discuss.

Suppose we have an input image $I$. The methodology introduces a set of $N$ masks $\{M_i\}_{i=1}^N$ indicating regions of interest in the image. The goal is to extract $N$ textual embeddings $\{v_i\}_{i=1}^N$, referred to as handles, such that each $v_i$ represents the concept in mask $M_i$ and that these handles can be combined in various textual prompts to generate new images or novel combinations of the concepts.

During **token optimization**, the model weights are frozen and the textual embeddings $v_i$ are optimized to reconstruct the input image $I$ using a masked diffusion loss:

$$L_{\text{rec}} = \mathbb{E}_{z,t,\epsilon\sim\mathcal{N}(0,1)}\Big[\big\|\epsilon \odot M_s - \epsilon_\theta(z_t, t, p_s) \odot M_s\big\|_2^2\Big], \tag{4.5}$$

where $M_s = \bigcup_{i\in s} M_i$ is the combined mask for the selected concepts. For example, for an image representing a dog and a cat, we would generate masks for dog and cat respectively. During training, we would denoise the entire image, but calculate the loss only for how well the noise is estimated at masked region.

During **Weights Optimization**, both the handles $\{v_i\}$ and model weights $\theta$ are fine-tuned with a smaller learning rate to improve fidelity without overfitting.

To ensure that the model can synthesize combinations of concepts, a union sampling strategy is employed. At each training step, a random subset $s \subseteq \{1, \ldots, N\}$ of concepts is selected. A prompt is constructed, e.g., ‘‘`a photo of [`$v_1$`] and [`$v_2$`],`’’ and the corresponding combined mask $M_s$ is used for the diffusion loss. This ensures that the handles are jointly trained to generate individual concepts and their combinations.

Another novel feature of the work is the introduction of **cross-attention loss** for disentanglement:

$$L_{\text{attn}} = \mathbb{E}_{z,t}\left[\left\|\text{CA}_\theta(v_i, z_t) - M_i\right\|_2^2\right] \tag{4.6}$$

where $\text{CA}_\theta(v_i, z_t)$ is Cross-attention map between handle $v_i$ and noisy latent $z_t$. The loss measures how well the cross-attention maps align with their corresponding masks, ensuring that the model is focusing on the specific object of interest. The total loss becomes:

$$L_{\text{total}} = L_{\text{rec}} + \lambda_{\text{attn}} L_{\text{attn}} \tag{4.7}$$

where $\lambda_{\text{attn}}$ is a weighting factor.

### 4.4.4 Reversion

While the methods discussed so far aim to capture objects, the work [23] extends textual inversion to encoder higher-order concepts such as relationships between objects. The work design a novel relation-steering contrastive learning scheme to steer the relation prompt towards a relation-dense region in the text embedding space. They use a set of basis prepositions as positive samples to pull the embedding into sparsely activated regions while treating words from other parts of speech (e.g., nouns, adjectives) in text descriptions as negative samples. This approach helps disentangle semantics related to object appearances. To further emphasize object interactions, the authors propose a relation-focal importance sampling strategy, which constrains the optimization process to prioritize high-level interactions over low-level details.

Appearance inversion focuses on inverting low-level features of a specific entity, thus the commonly used pixel level reconstruction loss is sufficient to learn a prompt that captures the shared information in exemplar images. In contrast, relation is a high-level visual concept. A pixelwise loss alone cannot accurately extract the target relation. Some linguistic priors need to be introduced to represent relations.

The authors present the **preposition prior**, a language-based prior that steers the relation prompt towards a relation-dense region in the text embedding space. This prior is motivated by a well-acknowledged premise and two interesting observations on natural language. The first premise of the authors is that **Prepositions describe relations**. In natural language, prepositions are words that express the relation between elements in a sentence. To back this premise, they note the **POS clustering** in the CLIP space. Embeddings are generally clustered according to their Part-of-Speech (POS) labels in CLIP. This observation motivates the authors to steer the relation prompt $\langle R \rangle$ towards the preposition subspace (i.e., the red region in Figure below)

**INSERT FIGURE**

As illustrated in the figure below, the feature similarity between a real-world relationship and prepositional words exhibits a sparse distribution. The prepositions that are activated tend to align closely with the semantic meaning of the given relationship. For instance, in the case of the relationship "swinging," prepositions such as "underneath," "down," "beneath," and "aboard" are sparsely activated, collectively capturing the essence of the "swinging" interaction. This **second observation** highlights that only a specific subset of prepositions should be activated during the optimization process, which forms the basis of our noise-resistant design.

Similar to pseudo-word prompt, a relation prompt $\langle R \rangle$ is optimized using the reconstruction loss:

$$\langle R \rangle = \text{argmin}_{\langle r \rangle} \mathbb{E}\left[\|\varepsilon - \varepsilon_\theta\left(\vec{x}_t, \tau_\theta(\vec{c})\right)\|^2\right] \tag{4.8}$$

As the above discussion highlights, the above loss mainly focuses on pixel-level reconstruction rather than visual relation. In order to learn the more general concept of "relations" between objects, the authors utilize the embedding space of clip. They define propositions as positive samples and other POS' words (Nouns, adjectives) as negative samples to construct the following loss:

$$L_{\text{pre}} = -\log \frac{\exp\left(\frac{R^T \cdot P_i}{r}\right)}{\exp\left(\frac{R^T \cdot P_i}{r}\right) + \sum_{k=1}^{K} \exp\left(\frac{R^T \cdot N_i}{r}\right)} \tag{4.9}$$

Furthermore, only a small set of propositions should be considered as true positives. Thus, 4.9 needs to be defied as following:

$$L_{\text{steer}} = -\log \frac{\sum_{l=1}^{L} \exp\left(\frac{R^T \cdot P_i^l}{r}\right)}{\exp\left(\frac{R^T \cdot P_i}{r}\right) + \sum_{k=1}^{K} \exp\left(\frac{R^T \cdot N_i}{r}\right)} \tag{4.10}$$

Where $P_i = \left\{P_i^1, \ldots P_i^L\right\}$ refers to positive samples randomly drawn from a set of basis prepositions and $N_i = \left\{N_i^1, \ldots N_i^M\right\}$ refers to the improved negative samples. The above is the standard InfoICE loss used in contrastive learning settings.

Besides introducing the **relation-steering loss**, the authors introduce **relational-focal importance sampling**. To understand this, keep in mind that high-level semantics (like object relations) appear earlier in the denoising process, while finer details (like textures) emerge later. Since the goal of the model is to capture relations (a high-level concept) rather than details, focusing optimization on early stages of denoising

(where high-level semantics appear) is crucial.

Normally, the timestep t in diffusion models is sampled uniformly from all possible timesteps. However, to emphasize high-level relations, the paper proposes a skewed sampling strategy where larger timesteps (closer to the initial noise, where high-level semantics are more prominent) are sampled with higher probability. This ensures that the model learns more about object relations rather than focusing on low-level pixel details.

The paper defines the sampling function:

$$f(t) = \frac{1}{T}\left(1 - \alpha \cos\left(\frac{\pi t}{T}\right)\right) \tag{4.11}$$

Where T is the total number of timesteps and $\alpha \rightarrow [0, 1]$ controls the skewness of the distribution. Thus, we have the following sampling function:

$$L_{\text{noise}} = \mathbb{E}_{t\sim f}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{x}_t, \tau_\theta(\vec{c})\right)\right\|^2\right] \tag{4.12}$$

This loss encourages the model to prioritize learning relations rather than focusing on reconstructing fine details. The final optimization objective of the ReVersion framework is a weighted combination of the steering loss (which guides the model towards learning object relations) and the denoising loss (which captures high-level semantics using importance sampling):

$$\langle R\rangle = \text{argmin}_{\langle r\rangle}\left(\lambda_{\text{steer}}\, L_{\text{steer}} + \lambda_{\text{denoise}}\, L_{\text{denoise}}\right) \tag{4.13}$$

To understand the framework of reversion, we carry out an extensive example. Suppose the input text is "A cat is sitting on a chair." The relation words in this prompt are ['on', 'above', 'below'] and Stop Words are ['a', 'is', 'the', 'of', 'in', 'at', 'by']. We would use the following as the Placeholder Token *. Say we tokenize the input sentence "A cat is sitting on a chair.". After tokenization, we might get the token IDs assocaited with each word with the addition of [BOS] and [EOS] tokens. For our example, on is a relation word, a and is are stop words and other tokens like cat, sitting, and chair carry significant meaning. These tokens are converted into embeddings. Next, the authors create a stop mask that will identify the stop words, relation words, and special tokens. For our example, the words "cat", "red" and "chair" would be the negative samples in the contrastive loss calculations, stop-words would be removed. The positive samples are selected from the *relation_words* defined before hand. In this case, we might randomly sample one or more relation words, such as 'on', 'above', or 'below', and create embeddings for them.

## 4.5 Encoder-Based Fast-Tuning

The work [16] lives up to the promise that the authors made in their initial paper [15] of personalizing the diffusion model using a single image $\mathcal{J}$ rather than a collection of images. To understand their approach, let us first go through the normal personalization scheme. Usually, we employ the VQ-VAE encoder to map $\mathcal{J}$ to a latent $\vec{z}_0$ which is then sent to a U-NeT model that iteratively learns to map $\vec{z}_0$ to $\vec{z}_T$. At every point, the reconstruction is conditioned on the pseudovector $\langle\vec{d}\rangle$. The loss reads as following: $L_{LDM} = \mathbb{E}\left[\|\varepsilon - \varepsilon_\theta\left(\vec{z}_t, t, c_\theta\left(\langle\varepsilon_c\rangle\right)\right)\|^2\right]$

Now in this case, $\langle\vec{d}\rangle$ is generated by mapping the placeholder "*" to the CLIP space by tokenization and then continually improved by backpropagating using the above loss. In this paper, however, the authors first learn an encoder $E(\mathcal{J})$ that directly takes the generated image $\mathcal{J}$ to the clip-embedding space:

$$E(\mathcal{J}) \rightarrow \text{ CLIP-Embedding Space } \rightarrow \langle\vec{d}\rangle \tag{4.14}$$

Thus, encoder automatically map the image to the embedding space, providing an initial guess for the pseudovector $\langle\vec{d}\rangle$. With an encoder, we can bypass the iterative optimization process which traditionally requires multiple gradient updates to fine-tune the pseudovector. Thus, an encoder provides a **pre-learned mapping that reduces the number of required optimization steps**, or in some cases, removes the need for additional optimization entirely.

Therefore, if we can learn an encoder $E(\mathcal{J})$ that provides a good guess for $\langle\vec{d}\rangle$, we would have a significant speedup. Now, in order to make sense of what a good encoder should do, the authors explore what a good personalized embedding $\langle\vec{d}\rangle$ should look like. How well $\langle\vec{d}\rangle$ is optimized over depends on two things - distortion vs editability:

- If the latent code is too close to the real word embeddings (representing common objects or words), the concept might not be unique enough, leading to poor reconstruction.
- If the latent code is too far from the real word embeddings, the model might accurately reconstruct the concept but lose editability-meaning it will not generalize well to different contexts or prompts.

To provide an initial guess of $\langle\vec{d}\rangle$, the authors think in terms of a previous work in GANs. It suggests that the initial inversion constrains $\langle\vec{d}\rangle$ to an editable region of the latent space, at the cost of providing only an approximate match for the concept. We can then approximate that region by fine-tuning the generator. For example, suppose

you have a novel cat "Rosy". We would first aim to map it to the "cat" category in the word-embedding, and only then would we optimize our diffusion model to approximate from "cat" to "Rosy":

$$\langle \vec{d} \rangle = \langle \vec{c} \rangle + s \cdot E\left(I_c\right) \tag{4.15}$$

Where E is our encoder, $\langle \vec{c} \rangle$ is the pre-trained model's embedding for the domain's coarse descriptor, and $s$ is a scaling factor which we empirically set to 0.1 . They further introduce a regularization term for the encoder:

$$\left\| E\left(I_c\right) \right\|_2^2 \tag{4.16}$$

The term $\left\| E\left(I_c\right) \right\|_2^2$ ensures that the encoder learns simpler, more generalizable representations of the concept, avoiding overfitting to specific details of the input image.

As for the encoder $E\left(I_c\right)$, the authors use CLIP visual encoder to directly map $I_c$. The encoder extracts feature from the pre-trained OpenCLIP ViT-H/14 model:

$$E\left(I_c\right) = f_{\text{clip}}\ \left(I_c\right) + \ldots \tag{4.17}$$

The ... indicates the need for us to incorporate how the encoder learn to estimate the noisy latents from $\vec{z}_0$ to $\vec{z}_T$. When denoising from $\vec{z}_T$ to $\vec{z}_0$, they argue that mapping the latent back to the image incurs a significant cost in both memory and time. Thus, instead of carrying that out, at layer of the U-NeT, they extract the pooled features $\left\{ \vec{f}_1, \vec{f}_2, \ldots \vec{f}_6 \right\}$. Thus, the encoder becomes:

$$E\left(I_c\right) = E\left( f_t^{(l)} \otimes f_{clip}\left(I_c\right) \right) \tag{4.18}$$

To understand [**?**], let us consider $f_{\text{clip}}\ \left(I_c\right)$ more closely. The ViT divides the images into patches, embed them, and process the sequence of patches using a transoformer model. Now, ViT has multiple layers, 12, 16, 24 depending on the model. Instead of taking features from each and every layer (which can be redundant or too detailed), the authors decided to extract features from every second layer. Furthermore, they only extract the [CLS token]. This means if you have a model with 12 layers, you would extract the [CLS] token features from layers 2, 4, 6, 8, 10, and 12.

$$F_l = CLIP_l\left(I_c\right)_{CLS}$$

Each extracted feature is passed through a linear layer:

$$h_l = W_l F_l + b_l$$

The transformed feature vectors $h_l$ are then averaged across all the extracted layers, which forms a hierarchical representation. If there are n layers being used, the pooled feature can be written as:

$$h_{\text{pooled}} = \frac{1}{n} \sum_{l=1}^{n} h_l$$

The pooled features are passed through a LeakyReLU activation function:

$$h_{\text{activated}} = \text{LeakyReLU}\left(h_{\text{pooled}}\right)$$

After activation, the resulting features are passed through another linear layer to predict the embedding offset $E\left(\mathcal{J}_c\right)$ where $\mathcal{J}_c$ is the input concept image:

$$E\left(\mathcal{J}_c\right) = W_{\text{final}}\, h_{\text{activated}} + b_{\text{final}} + \ldots$$

Incorporating the features from the diffusion model, we have the final encoding off-set as following:

$$E\left(\mathcal{J}_c\right) = W_{\text{final}}\, h_{\text{activated}} + b_{\text{final}} \otimes f_t^{(l)} \tag{4.19}$$

In total, we have:

$$\langle \vec{d} \rangle_t = \langle \vec{c} \rangle + E\left(W_{\text{final}}\, h_{\text{activated}} + b_{\text{final}} \otimes f_t^{(l)}\right) \rightarrow \tag{16}$$

This word-embedding encoder predicts new code in the diffusion model's embedding space which best describes the input concept. Notice that during the time of training, this embedding is dynamic because of the features $f_t^{(l)}$ change at each iteration. Thus for each time step, we have $\langle \vec{d} \rangle_t$

**Weight-Set Optimization**

While the encoder is highly effective at creating a concept embedding, it doesn't directly control how the intermediate layers (especially the attention layers) of the diffusion model apply this new embedding to the image generation process. To mitigate these shortcomings, the authors introduce weight-offsets to optimize their model. Now, the authors carried out the normal textual inversion process, and they found that cross and self-attention layers undergo the highest change during tuning. Thus, these layers play the most crucial part in the tuning effort. Keeping this in mind, they modify three

attention projection matrices - $W_Q, W_K$ and $W_v$. The weight offsets for the attention matrices are learned using an update rule:

$$W^i_{q,k,v} = W^i_{q,k,v,0} \cdot \left(1 + \Delta W^i_{q,k,v}\right) \tag{4.20}$$

Here, $W^i_{q,k,v,0}$ are the original pre-trained weights, and $\Delta W^i_{q,k,v}$ are the learned offsets. The goal is to learn the smallest possible perturbations to these attention weights that still enable the model to incorporate new concepts.

**Brief Overview of LoRA** In large-scale models, the parameter matrices in neural networks (such as weights in attention layers) are typically high-dimensional and dense. LoRA assumes that the updates required for fine-tuning the model can be captured in a low-rank subspace of the parameter matrices. The key idea is to restrict the updates to this low-rank space, thus greatly reducing the number of trainable parameters. Recall that **the rank of a matrix** is the number of linearly independent rows or columns it has. In LoRA, a low-rank update means that instead of allowing full-rank changes to the model's weight matrices, we approximate the update using a low-rank decomposition. Assume you have a weight matrix $W_0$ in the model that you want to update. Instead of directly fine-tuning $W_0$, LoRA introduces two smaller matrices, A , and B such that: $W = W_0 + \Delta W = W_0 + \mathrm{BA}$. In this representation, A is a low-rank matrix of dimension $r \times \mathrm{d}$ where r is much small than original matrix. On the other hand, B is a low-rank matrix of dimension $d \times \mathrm{r}$ where d is the original dimension of $W_0$

Instead of directly learning the offsets $\Delta W^i_{q,k,v}$ the offsets are regularized through a neural network to ensure smoothness and avoid overfitting:

$$\Delta W^i_{q,k,v} = \mathrm{Linear}\left(v_x \cdot v_y\right) \tag{4.21}$$

Where $v_x$ and $v_y$ are linear projections of a learned initial parameter vector $v_0$. During pre-training, the encoder and weight offsets are trained on large datasets using a combination of a diffusion loss and the regularization loss:

$$\begin{aligned} L &= L_{\text{diff}} + \lambda_r L_{reg} \\ L &= L_{\text{diff}} + \lambda_r \left\| E\left(I_c\right)\right\|_2^2 \end{aligned} \tag{4.22}$$

Where $\left\| E\left(I_c\right)\right\|_2^2 = E\left(W_{\text{final}}\, h_{\text{activated}} + b_{\text{final}} \otimes f_t^{(l)}\right)$

While the following method can achieve high-fidelity personalization with short

training times, the work is not without limitations. First, the encoders rely on learning to generalize from large datasets that represent the coarse target class. This reliance means that the method is applicable only to classes where such large datasets exist. In practice, this includes categories such as faces and artistic styles, which are the primary use cases for personalization. However, this limitation restricts the method's applicability to rare or one-of-a-kind objects. For concepts within nearby domains—such as dogs personalized using a model trained on cats—the method still produces high-fidelity results. In contrast, for more distant domains, such as a wooden toy, the method fails to capture concept-specific details accurately.

A further limitation of this work lies in its requirement for inference-time tuning. Although the additional synthesis time is relatively minor, the approach requires that the inference machine be capable of performing model tuning. Moreover, the need to tune both the encoder and text-to-image models simultaneously increases the memory requirements compared to direct fine-tuning methods. These constraints may limit the method's usability in resource-constrained environments or applications with non-standard datasets.

# 5 Semantic Correspondence and Semantic Segmentation

## 5.1 TO-Do and Ideas

1. Can we use Diffusion models to make predictions on constitutionality? For example, given a "leg", classify whether the leg belongs to a 'human', 'cat', or a 'dog'. That is, the task of **using diffusion models to predict objects from their parts**
2. Using diffusion models for conditional generation where an existing image or video can be edited to modify the nature of the interaction (e.g., changing how a person interacts with an object). This could be useful in creative applications or even training simulations.
3. Write an intro explaining how self-attention naturally aids segmentation tasks and cross-attention aids correspondences task
4. I think I understand the theoretical reason behind why "catastrophic forgetting" occurs. I was comparing the words in Diffusion Model's Clip tokenizer and playing with them and realized that for words like "cat" and "dog", the similarity is 1 for the word-embedding. The Clip's Encoder space is unable to distinguish between the two
5. Incorporate more information in DiffewS

## 5.2 Introduction to Semantic Segmentation

**Semantic image segmentation** aims to assign a corresponding class label to every pixel in an image. Due to the pixel-level granularity of predictions, this task is often referred to as **dense prediction**. It is important to note that semantic segmentation does not differentiate between instances of the same class; it focuses solely on categorizing each pixel. For example, if an image contains two objects belonging to the same category, the resulting segmentation map does not inherently distinguish these as separate entities. This distinction is addressed by a different type of model, known as **instance segmentation**, which separates individual instances of the same class.

Segmentation models are instrumental in a wide range of applications including:

- **Autonomous Vehicles:** These models enable vehicles to perceive and interpret their surroundings, a critical capability for ensuring the safe integration of self-driving cars into existing road systems.
- **Medical Image Diagnostics:** Segmentation models enhance the efficiency and accuracy of radiological analysis, significantly reducing the time required for diagnostic assessments by augmenting the work of medical professionals.

**Representing the Task:** Given an input I of size W × H × C, the task is to produce a segmentation map S of the same size where each pixel $S(x, y)$ corresponds to a class label c from a set of predefined classes C = $\{c_1, c_2 \dots c_N\}$ where N is the number of classes. Then, the segmentation map S can be defined as:

$$\mathrm{S} : \{1, 2 \dots \ \mathrm{W}\} \times \{1, 2, \dots \mathrm{H}\} \rightarrow \{\mathrm{c}_1, \mathrm{c}_2, \dots \mathrm{c}_\mathrm{N}\} \tag{5.1}$$

For each pixel (x, y) in the image $I$ , the task is to predict the class label $S(x, y)$ that best describes the object or region in which the pixel belongs. To understand the framework, imagine we have a simple 4 × 4 pixel image and our task is to perform semantic segmentation on it for a scenario where we have distinct classes: Background ($c_1$), Cat ($c_2$) and Dog ($c_3$). For simplicity, lets consider the following ( 4 × 4 ) image representation where each letter represents a pixel belonging to a different class:

$$\text{Ground Truth Segmentation Map } \mathrm{S}^*(\mathrm{x}, \mathrm{y}) = \begin{bmatrix} B & B & C & C \\ B & D & D & C \\ B & D & D & C \\ B & B & C & C \end{bmatrix}$$

Where B represents the Background, C represents the Cat and D represents the Dog. We call S* as the ground truth segmentation map. The goal of semantic segmentation

is to assign each pixel in the image a label corresponding to one of these classes. After processing this image through a semantic segmentation model, we aim to get a segmentation map $S$ that mirrors the layout of the ground truth segmentation map in terms of class assignments. The segmentation map S for the given example could be represented as a matrix of the same dimension as the image, where each element is the class label predicted for the corresponding pixel:

$$\text{Segmentation map } S(x,y) = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 3 & 3 & 2 \\ 1 & 3 & 3 & 2 \\ 1 & 1 & 2 & 2 \end{bmatrix}$$

Here, the numbers $(1, 2, 3)$ correspond to Background, Cat and Dog. So $S(x, y)$ gives us the class of the pixel at position ( x, y ) in the image. After we have the segmentation map S(x, y ), we compare it with the ground truth segmentation map $S^*$. A common choice is the cross-entropy loss. For a single pixel, it is defined as following:

$$L_{CE}\left(S(x,y), S^*(x,y)\right) = -\log\left(\frac{\exp\left(S(x,y)\left[c^*\right]\right)}{\sum_{c\in C}\exp(S(x,y)[c])}\right) \tag{5.2}$$

Where $S(x, y)[c]$ is the predicted probability of pixel (x, y) belonging to class c , and $c^*$ is the actual class label for that pixel in the ground truth segmentation map $S^*$. The total loss for the image is the sum of the pixel-wise losses:

$$L = \sum_{x=1}^{W}\sum_{y=1}^{H} L_{CE}\left(S(x,y), S^*(x,y)\right) \tag{5.3}$$

Note that the above approach assumes one-hot encoding of class labels. For example, consider the image above. The total classes we have are {person, purse, sidewalk, building.

Equation 5.3 highlights that training a semantic segmentation model can be computationally expensive when considering pixel-wise loss since the loss is calculated and summed over all pixels in the image for each training instance. One way to mitigate this is through a method know as the **"center pixel prediction"**. In this approach, instead of predicting the class label for each pixel individually, the model focuses on predicting the class label for the center pixel of a small region or patch around each pixel. This center pixel is typically chosen to be the pixel at the center of the receptive field of the convolutional neural network (CNN). The predicted class label for this center pixel is then assigned to all pixels in the corresponding patch.

The cross-entropy loss calculates predictions for each pixel vector independently

and averages the results across all pixels. This approach essentially assigns equal importance to each pixel during training. However, if there is an imbalance in class representation within an image, this method can lead to biased training dominated by the most common class. To address this issue, Long et al. (FCN paper) propose weighting the loss for each output channel to balance class representation in the dataset.

Ronneberger et al. (U-Net paper) introduce a different strategy that adjusts the loss weight for each pixel. Their method assigns greater weight to pixels at the borders of segmented objects, which helps their U-Net model achieve better segmentation of cells in biomedical images. This technique allows individual cells to be easily identified in binary segmentation maps, even when the cells are closely packed.

Another widely used loss function for image segmentation tasks is derived from the Dice coefficient, a metric that measures the overlap between two sets. The Dice coefficient ranges from 0 to 1, with a value of 1 indicating complete overlap. Originally developed for binary data, the Dice coefficient can be computed as:

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \tag{5.4}$$

**Encoder-Decoder Structures for Segmentation**: As discussed, the process of associating pixel-level information with image patches can be computationally expensive. To address this challenge, convolutional neural networks are often employed. Although a simple convolutional neural network can be used for segmentation, the computational burden of pixel-level operations remains substantial. To reduce this burden, one widely adopted approach for image segmentation involves utilizing an **encoder-decoder architecture**. In this design, the encoder module reduces the spatial resolution of the input image, creating low-resolution feature mappings that efficiently capture discriminative information for classification. The decoder module then upsamples these feature representations to reconstruct a full-resolution segmentation map. In CNN architectures, transposed convolutions are utilized.

A limitation of encoder-decoder structure is that the encoder module significantly reduces the spatial resolution of the input, making it challenging for the decoder module to produce fine-grained segmentations. This issue highlights a fundamental trade-off between semantic and spatial information. While global features resolve *what* is present in the image, local features resolve *where* objects are located. To address this, the authors of Ronneberger et al. introduced a technique that gradually upsamples the encoded representation in stages, incorporating **skip connections** from earlier layers. These skip connections merge fine-grained features from the encoder with coarse-grained features from the decoder. This enables the model to make localized

predictions that align with global context.

**Zero-shot semantic segmentation:** Let the class space be $C = C_{\text{seen}} \cup C_{\text{unseen}}$ where $C_{\text{seen}}$ are the classes seen during training and $C_{\text{unseen}}$ are classes unseen during training. The auxiliary information $A(c)$ for each class $c$ provides semantic representations that link seen and unseen classes. Formally, the task is to predict the pixel-wise segmentation mask $\hat{M}$ for an image $I$, assigning labels to pixels in $C_{\text{unseen}}$, without directly observing labeled data for $C_{\text{unseen}}$ during training.

The methodology often utilizes a feature extractor and additionally, there is a function $g(c)$ which encodes the auxiliary information of each class $c$. Given these, we define the **Compatibility Function** $\phi(F, g(c))$, which measures the compatibility between image features and class embeddings, often defined as:

$$\phi(F, g(c)) = F(x, y) \cdot g(c)$$

During inference, the segmentation mask $\hat{M}$ is predicted by assigning each pixel the label of the most compatible class:

$$\hat{M}(x, y) = \underset{c \in C_{\text{unseen}}}{\operatorname{argmax}} \, \phi(F(x, y), g(c))$$

where $F(x, y)$ represents the feature at pixel location $(x, y)$.

A simple example of zero-shot segmentation is segmenting animals in an image where the model was trained to recognize only cats and dogs but is tasked with identifying a horse (an unseen class). The model uses semantic information like textual descriptions (e.g., "a horse has four legs and a tail") to generalize and segment the horse in the image without ever having seen labeled examples of horses during training

**Few-shot semantic segmentation:** Few-shot segmentation involves segmenting objects in an image where only a few annotated examples (support samples) of the target class are available for training. Let the support set be $S = (I_s^i, M_s^i)_{i=1}^K$, where $I_s^i$ is the $i$-th support image, and $M_s^i$ is its corresponding segmentation mask for the target class. Given a query image $I_q$, the goal is to predict a segmentation mask $M_q$ that identifies regions in $I_q$ corresponding to the target class in $S$.
The methodology utilizes a feature extractor for both support and query images. A **Support-Query Relationship Function**

$$\phi(S, I_q)$$

is defined to measure the relationship between the support set and the query image, allowing the model to adapt to the target class with limited examples.

A simple example of few-shot segmentation is segmenting a dog in a query image when only one support image of a dog with its segmentation mask is provided. The model learns from this single annotated example to identify and segment the dog in the query image.
**Unsupervised semantic segmentation:** Unsupervised semantic segmentation involves segmenting an image into semantically meaningful regions without the use of any labeled data. Let an input image be $I$, and the goal is to predict a segmentation mask $\hat{M}$, where each pixel is assigned to a cluster that represents a distinct semantic region, without any prior supervision or annotations.
The methodology often employs unsupervised feature extraction techniques, such as clustering in the feature space, to group pixels with similar semantic meanings. A **Clustering Function** $\phi(F)$ is used, where $F$ represents the extracted features from the input image $I$, to partition the image into clusters corresponding to semantic regions.
A simple example of unsupervised semantic segmentation is segmenting an image of a natural scene into regions such as sky, water, and land, without any labeled data. The model clusters pixels with similar visual patterns, such as blue for the sky and water or green for the land, to generate a segmentation mask.

## 5.3 Segmentation using Statistical Physics

There are a few ideas from statistical physics that can be utilized for segmentation purposes. Both the **Ising model** and the **Potts model** can be used for image segmentation. The Potts model is often used because it allows for multiple possible labels

(or states) at each pixel, corresponding to different segments in the image. The Ising model can also be used for binary segmentation tasks (where there are only two possible labels, such as "foreground" and "background").

**Ising Model for Binary Segmentation:** In the Ising model, each site (or pixel in the case of image segmentation) can take one of two possible states, typically denoted as $s_i \in \{-1, +1\}$. The energy function for the Ising Model is:

$$H(s) = -J \sum_{<i,j>} s_i s_i - \sum_i h_i s_i \tag{5.5}$$

Where $h_i$ is the local external field acting on pixel $i$. We can think of the terms in equation (4) as consisting of the following two terms:

1. Unary term $\sum_i h_i s_i$ : This term represents how well each pixel fits into a particular class (based on the pixel intensity or other features).

2. Pairwise term $\sum_{<i,j>} s_i s_i$ : This term encodes the interaction between neighboring pixels, penalizing configurations where neighboring pixels have different labels (to enforce smoothness).

In binary segmentation, the goal is to segment the image into two regions, $\mathcal{R}_1$ (foreground) and $\mathcal{R}_2$ (background). The Ising model provides a natural framework for this problem by associating each pixel with a spin variable $s_i$ where,

- $s_i = +1$ represents that pixel i belongs to the foreground
- $s_j = +1$ represents that pixel j belongs to the foreground

The segmentation task can be cast as finding the configuration of spins $s = \{s_1, s_2 \ldots, s_N\}$ that minimizes the following energy function:

$$E(s) = \sum_i \psi_u \left(s_i, I_i\right) + \sum_{<i,j>} \psi_p \left(s_i, s_j\right)$$

Where $\psi_u \left(s_i, I_i\right)$ is the unary potential (or data term) that encodes how well pixel i's intensity $I_i$ fits with label $s_i$ (foreground or background). $\psi_p \left(s_i, s_j\right)$ is the pairwise potential that encodes the interaction between neighboring pixels $i$ and $j$, encouraging neighboring pixels to have the same label.

The unary potential is typically derived from the likelihood of pixel i's intensity given the foreground or background distribution. Suppose the pixel intensities are modeled as Gaussian distributions for the foreground and background:

$$P\left(I_i \mid s_i = +1\right) = \mathcal{N}\left(I_i; \mu_1, \sigma_1^2\right), P\left(I_i \mid s_i = -1\right) = \mathcal{N}\left(I_i; \mu_2, \sigma_2^2\right)$$

The unary term $\psi_u\left(s_i, I_i\right)$ can then be defined as the negative log-likelihood:

$$\psi_u\left(s_i, I_i\right) = \begin{cases} -\log P\left(I_i \mid s_i = +1\right) & = \frac{(I_i - \mu_1)^2}{2\sigma_1^2}, \text{ if } s_i = +1, \\ -\log P\left(I_i \mid s_i = -1\right) & = \frac{(I_i - \mu_2)^2}{2\sigma_2^2}, \text{ if } s_i = -1 \end{cases} \tag{5.6}$$

This unary term encourages pixels with intensities close to $\mu_1$ to be labeled as foreground and those close to $\mu_2$ to be labeled as background $s_i = -1$

The pairwise potential penalizes neighboring pixels that have different labels (i.e., different spin states) to enforce spatial smoothness. The simplest form of the pairwise term is:

$$\psi_p\left(s_i, s_j\right) = J s_i s_j$$

Where J > 0 encourages neighboring pixels i and j to have the same label (spin state). This term is inspired by the original Ising model, where spins prefer to align with their neighbors to minimize the energy.

The total energy is:

$$E(s) = \sum_i \psi_u\left(s_i, I_i\right) + \sum_{<i,j>} J s_i s_j \tag{5.7}$$

The goal is to minimize the energy $E(s)$ to find the optimal segmentation. This can be a challenging optimization problem because the energy function involves interactions between neighboring pixels (pairwise terms).

**Potts Model for Segmentation** : The Potts model extends the Ising model for multi-class segmentation. The energy function for the Potts model is defined as:

$$E(Y) = \sum_{i,j} \mathbb{I}\left(Y_i \neq Y_j\right) \tag{5.8}$$

Where $Y_i$ and $Y_j$ are the labels of neighboring sites and $\mathbb{I}\left(Y_i \neq Y_j\right)$ is an indicator function that penalizes neighboring sites if they have different labels. In some sense, the Potts model is the generalization of the Ising Model. The Ising model energy is defined as following in case of no external magnetic field:

$$H = -J \sum_{\langle i,j \rangle} s_i s_j \tag{5.9}$$

Where $s_i \in \{-1, +1\}$ and we know that the Ising model prefers configurations where the spins are aligned. The Potts model generalizes the Ising model by allowing more than two possible spin states at each site. It is used to model systems where each site (or particle) can be in one of **q** discrete states. The Hamiltonian (energy function) for the Potts model is given by:

$$H = -J \sum_{\langle i,j \rangle} \delta(s_i, s_j)$$

For q = 2, the Potts model is equivalent to the Ising model but for $q > 2$, the Potts model can exhibit a first-order (discontinuous) phase transition, where the transition between ordered and disordered states occurs abruptly.

**Graph Optimization for Solving the Ising Hamiltonian:** The Ising Model (5.7) is solved using graph optimization. To carry this out, we first create a graph in which each pixel i in the image corresponds to a node in the graph. Two additional nodes are added: the source node S (representing the foreground) and the sink node T (representing the background).

**Insert Figure Here!**

There are two types of edges in the graph:

1. **T-Links**: These connect each pixel node i to either the source node S or the sink node T . The weight of these edges encode the unary terms $\psi_u(s_i)$ which represents the cost of assigning pixel $i$ to the foreground or background. For each pixel i, two edges are added:

   (a) An edge from i to the source with weight $w(i, S) = \psi_u(s_i = 1)$ (cost of assigning i to the foreground)

   (b) An edge from i to the sink with weight $w(i, T) = \psi_u(s_i = 0)$ (cost of assigning i to the background)

2. **N-links**: These connect neighboring pixel nodes i and j in the terms. The weights of these edges encode the pairwise terms $\psi_p(s_i, s_j)$ which represent the cost of assigning different labels to neighboring pixels.

The goal of the segmentation problem is to minimize the energy function $E(s)$ by cutting the graph in such a way that the total cost (or energy) is minimized. The segmentation problem is solved by finding the minimum cut that separates the graph into two disjoint sets:

- One set containing the source node $S$ and the pixels labeled as foreground.
- The other set containing the sink node T and the pixels labeled as background.

The cut is defined as a set of edges that, when removed, separates the source from the sink. The cost of the cut corresponds to the sum of the weights of the edges in the cut. More formally, Given a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ where V is the set of vertices including the source S and sink T and E
is the set of edges with associated weights, a cut C in the graph is a partition of the vertices into two disjoint sets:

- $\mathrm{S} \subseteq \mathrm{V}$ containing the source.
- $\mathrm{T} \subseteq \mathrm{V}$ containing the sink.

The cost of the cut $C$ is defined as the sum of the weights of the edges that cross the cut:

$$\mathrm{cost}(C) = \sum_{(i,j)\in C} w(i,j) \tag{5.10}$$

The minimum cut is the cut that has the smallest possible cost, i.e., the one that minimizes the total energy function $E(s)$ of the segmentation.

Unary Term Representation: The unary term $\psi_u(s_i)$ assigns each pixel to either the foreground or background based on the likelihood that the pixel belongs to that class (e.g., using intensity or color information). This is represented as the capacity of the T-links:

- $\mathrm{w(i, S)}$ = cost of assigning pixel i to the foreground
- $\mathrm{w(i, T)}$ = cost of assigning pixel $i$ to the background

Pairwise Term Representation: The pairwise term $\psi_p(s_i, s_j)$ encourages smoothness by penalizing neighboring pixels with different labels. This is represented as the capacity of the N links between neighboring pixels:

$\mathrm{w(i, j)} =$ cost of assigning different labels to neighboring pixels i and

This weight is often modeled as:

$$w(i,j) = \lambda \exp\left(-\frac{\|I_i - I_j\|^2}{2\sigma^2}\right) \tag{5.11}$$

Where $\|I_i - I_j\|^2$ is the squared difference in intensity or color between pixels i and j , and $\lambda$ controls the strength of the smoothness penalty. The Gaussian term ensures that neighboring pixels with similar intensities are more likely to be assigned the same label.

## 5.4 PACL

The paper "Open Vocabulary Semantic Segmentation with Patch Aligned Contrastive Learning" [28] introduces a method called Patch Aligned Contrastive Learning (PACL) for aligning image patches from a vision encoder with text tokens from a text encoder in models like CLIP. The approach is designed for **zero-shot semantic segmentation**, meaning it can segment images into classes not explicitly present in its training data without requiring segmentation annotations.

In traditional contrastive learning, such as in models like CLIP, the objective is to learn a similarity between entire images and their corresponding text descriptions, globally aligning the CLS token (a special token summarizing the entire image or text) from the image encoder and text encoder. This alignment helps the model understand whether a given image matches a description, but it works at the level of whole images, not specific parts of images (patches).

CLIP uses a contrastive loss to align image-text pairs. Each image and its corresponding text are represented by a global embedding. Specifically, the image encoder $f_v$ takes an image x and generates a CLS token, a single embedding representing the entire image:

$$f_v(x) \in \mathbb{R}^{D_v} \tag{5.12}$$

Where $D_v$ is the dimension of the embedding.
In PACL (Patch Aligned Contrastive Learning), the goal is to move beyond this global alignment and perform a finer, patch-level alignment. Here, instead of aligning the entire image and text globally, PACL aligns individual image patches (parts of the image) with the CLS token from the text encoder. This allows the model to identify which parts of the image correspond to specific text descriptions.

Instead of encoding the whole image into a single CLS token, the vision encoder in PACL breaks the image into patches and outputs a series of embeddings, one for each patch. So, instead of $f_v(x) \in \mathbb{R}^{D_v}$, PACL uses:

$$f_v(x) \in \mathbb{R}^{TxD_v} \tag{5.13}$$

Where T is the number of patches in the image. The text encoder still produces a global CLS token for the entire text y. PACL computes the cosine similarity between the text CLS token and each of the T patch embeddings. This gives T similarity values, one for each patch:

$$s(x,y) \in \mathbb{R}^T, s(x,y)_i = \frac{e_v\left(f_v(x)_i\right) \cdot e_t\left(f_t(y)\right)}{\left|e_v\left(f_v(x)_i\right)\right| \left|e_t\left(f_t(y)\right)\right|} \tag{5.14}$$

Once we have the similarity scores $s(x,y)_i$ for all patches in the image, PACL uses these scores to create weights for each patch. The weights are computed using the softmax function over all the patches:

$$a(x,y)_i = \frac{\exp\left(s(x,y)_i\right)}{\sum_{j=1}^{T} \exp\left(s(x,y)_j\right)} \tag{5.15}$$

To generate a global image representation, PACL aggregates the patch embeddings by taking a weighted sum where the weights are derived from the patch-text similarity scores $s(x,y)$ :

$$\hat{v} = \sum_{i=1}^{T} a(x,y)_i e_v\left(f_v(x)_i\right) \tag{5.16}$$

By summing the weighted embeddings of all patches, we get a single vector $\hat{v}$ that represents the most relevant parts of the image, considering the text description. In other words, instead of treating each patch equally, the model emphasizes the patches that are most relevant to the text, aggregating them into a single vector.

During training, PACL uses the weighted patch embedding $\hat{v}$ and aligns it with the text CLS token using a contrastive loss (InfoNCE). The training objective is to make sure that the weighted sum of patches from the correct image-text pair is more similar than any incorrect pair.

The InfoNCE loss is used, where for each correct image-text pair $(x_i, y_i)$, the similarity $\hat{\phi}(x,y)$ should be maximized, while the similarity for incorrect pairs $\hat{\phi}\left(x_i, y_i\right)$ should be minimized:

$$\widehat{\phi}(x,y) = \frac{\hat{v} \cdot e_t\left(f_t(x)_i\right)}{|\hat{v}| \cdot \left|e_t\left(f_t(y)\right)\right|} \tag{5.17}$$

Thus, the $L_{\text{InfonCE}}$ loss assumes the form:

$$L_{\text{InfoNCE}} = -\log \frac{\exp\left(\widehat{\phi}\left(x_i, y_i\right)\right)}{\sum_{j=1}^{k} \exp\left(\widehat{\phi}\left(x_j, y_j\right)\right)} \tag{5.18}$$

This contrastive loss encourages the model to learn the alignment between the relevant image patches and the text, resulting in a strong patch-text correspondence. At inference time, PACL can be used for zero-shot semantic segmentation. Given an image and a set of class labels (in text form), the model computes the similarity between each patch in the image and each class label. For each class, a segmentation mask is generated based on the similarity scores between the patches and the class text.

**Example:** Imagine you have an image of a park. The image has multiple objects: trees, people, and a dog. The text description could be something like: " A dog running in the park". You want the model to learn that the part of the image containing the dog corresponds to the word "dog" in the text, and the part of the image containing the trees corresponds to "park.". The first thing PACL does is divide the input image into patches. Let's assume the image is divided into 16 patches (though in practice, there are usually more). Each patch is a small portion of the image, say:

- Patch 1: Part of the sky
- Patch 2: Part of a tree
- Patch 3: Part of a dog
- ... and so on

So we have 16 patches $x_1, x_2 \ldots x_{16}$. Next, the image patches and the text description are encoded by the model. The vision encoder $f_v$ processes each image patch $x_i$ and generates an embedding (a vector) for each patch. So, you get 16 vectors, one for each patch:

$$f_v\left(x_i\right) \in \mathbb{R}^{D_v}, \text{ for each patch, i} = 1, 2, \ldots 16$$

The text encoder $f_t$ processes the text ("A dog running in the park") and generates a CLS token embedding, which is a vector summarizing the entire text:

$$f_t(y) \in \mathbb{R}^{D_t}$$

The key is that these embeddings for the image patches and the text are projected into the same shared space (through functions $e_v$ and $e_t$ ) so they can be compared. Note that these functions $e_v$ and $e_t$ are learned during training. Using these learned functions, we learn:

$$s(x,y)_i = \frac{e_v\left(f_v(x)_i\right) \cdot e_t\left(f_t(y)\right)}{\left|e_v\left(f_v(x)_i\right)\right| \left|e_t\left(f_t(y)\right)\right|}$$

This gives a similarity score $s(x,y)_i$ for each patch. For example, for the text description "A person is walking with a dog in the park." And 4 patches: 1. Sky, 2. A tree, 3. A person and 4. A dog, the goal is to compute how much each patch relates to the given text description, using the following steps:

- $s(x,y)_1$ (Sky and text): Suppose this patch of sky is not relevant to the text "A person is walking with a dog in the park." So, the similarity score might be low, e.g., $s(x,y)_1 = 0.1$.
- $s(x,y)_2$ (Tree and text): A tree in the park is somewhat relevant to the text, but it's not the main focus. So, the similarity score might be moderate, e.g., $s(x,y)_2 = 0.4$
- $s(x,y)_3$ (Person and text): Since the text describes a person walking, this patch should be highly relevant. Therefore, the similarity score might be high, e.g., $s(x,y)_3 = 0.8$
- $s(x,y)_4$ (Dog and text): The dog is central to the text description, so this patch will have a high similarity score, e.g., $s(x,y)_4 = 0.9$.

The final softmax weights $a(x,y)_i = \frac{\exp(s(x,y)_i)}{\sum_{j=1}^{T} \exp(s(x,y)_j)}$ represent how much attention or weight each patch should get in the final representation. It comes out as $[0.152, 0.205, 0.306, 0.338]$ = [Sky, Tree, Person, Dog]. Thus, the two most relevant features of interest comes out to be dog and person. The final step is to compute the weighted sum of the patch embeddings using these weights to give $\hat{v}$. This is a new vector that represents the image, but with more emphasis on the patches that are most relevant to the text (in this case, the dog and the person).

The model then uses contrastive learning to align this vector $\hat{v}$ with the text CLS embedding $e_t\left(f_t(y)\right)$.

Inference: This involves using the learned model to perform tasks like zero-shot semantic segmentation without explicit training for segmentation. The model is able to infer which parts of the image correspond to specific text labels (like "dog", "person") by aligning image patches with text descriptions. During inference, we are given:

- An image (e.g., a photo of a park with a dog, a person, and trees).
- A set of class labels in the form of text (e.g., ["dog", "person", "tree"]). These labels might describe objects you want to segment or recognize in the image.

The task is to predict which parts of the image correspond to each label, producing a segmentation mask or classifying each patch with the most likely label. As in training, the first step is to divide the input image into patches. Let's assume the image is divided into T patches, each patch representing a small portion of the image:

- Patch 1: Part of the sky.
- Patch 2: Part of a tree.
- Patch 3: Part of the person.
- Patch 4: Part of the dog.
- And so on, for all T patches in the image.

Each patch is passed through the vision encoder $f_v$ to generate a feature embedding for each patch. The vision encoder produces T embeddings, one for each patch: $f_v(x_i) \in \mathbb{R}^{D_v}$ where $D_v$ is the dimensionality of the patch embeddings. The set of text labels is passed through the text encoder $f_t$. Each label (e.g., "dog", "person", "tree") is encoded into an embedding $f_t(y_c) \in \mathbb{R}^{D_t}$. For each patch i in the image and each class label $y_c$, we compute the cosine similarity between the patch embedding $f_v(x)_i$ and the text embedding $f_t(y_c)$. This similarity tells us how well the patch corresponds to the class label. The similarity $s(x, y_c)_i$ for patch i and class $y_c$ is computed as:

$$s(x, y)_i = \frac{e_v(f_v(x)_i) \cdot e_t(f_t(y))}{|e_v(f_v(x)_i)| \, |e_t(f_t(y))|}$$

This produces a similarity score between each image patch and each class label. For example, if patch 4 contains the dog and the text label is "dog", the similarity score $s(x, \text{dog})_4$ would be high. Conversely, if patch 1 is part of the sky and the label is "dog", the similarity score $s(x, dog)_1$ would be low. For each patch, you compute a softmax over the class label similarity scores to assign each patch to the most likely class. The softmax is computed over all class labels C for each patch $a(x, y)_i = \frac{\exp(s(x,y)_i)}{\sum_{c=1}^{C} \exp(s(x,y)_j)}$. This softmax function converts the raw similarity

Once the softmax scores $a(x, y)_i$ are computed for all patches and class labels, each patch is assigned to the class with the highest probability. This can be used to generate a segmentation mask where each patch is labeled with the class that it most likely corresponds to.

For example:

- Patch 4 might be labeled "dog" $a(x, y)_4$ is high

- Patch 3 might be labeled "person".
- Patch 2 might be labeled "tree".

The resulting output is a segmentation map that tells you which parts of the image correspond to each object class.

## 5.5 PiCIE

The paper "PiCIE: Unsupervised Semantic Segmentation Using Invariance and Equivariance in Clustering" [11] presents a method for unsupervised semantic segmentation without the need for labeled data. The goal of this method is to discover and segment out high-level concepts (e.g., trees, sky, houses) from images without any human-provided annotations.

The key challenge in unsupervised semantic segmentation is how to cluster and label pixels in a way that reflects meaningful semantic objects. Traditional clustering methods are limited to simple, object-centric images, and fail when applied to complex, scene-centric datasets. PiCIE addresses this limitation by proposing a method that assigns cluster memberships to pixels while learning features that incorporate both invariance to photometric transformations (e.g., color changes) and equivariance to geometric transformations (e.g., cropping, flipping).

**Methodology:** PiCIE clusters pixels based on their feature representations. Let $\vec{x}_i$ represent an image from a dataset, and let $f_\theta(\vec{x}_i)$ be the feature representation of the image learned by a convolutional neural network ( CNN ) with parameters $\theta$. For each pixel p in the image, the CNN produces a feature vector:

$$f_\theta(\vec{x}_i) = \text{ Feature map for all pixels in the image}$$

PiCIE employs k-means clustering to group pixels based on their feature representations. Clustering is performed to discover semantic groups of pixels:

$$\min_{y,\mu} \sum_{i,p} \left\| f_\theta(x_i)[p] - \mu_{ip} \right\|^2$$

Where $y_{ip}$ is the cluster label of pixel p in image $x_i$ and $\mu_k$ is the centroid of the k -th cluster. The distance is typically the Euclidean distance in feature space.

The learning process is iterative and follows the standard k-means algorithm steps. Initially, the centroids $\mu_k$ for k $= 1, \ldots$ K are randomly initialized. For each pixel

p in image $x_i$, its feature vector $f_\theta(x_i)[p]$ is compared to the current centroid $\mu_k$. Each pixel is assigned to the cluster whose centroid is closest in terms of the distance metric (typically Euclidean or cosine distance). The assignment step is given by $y_{ip} = \arg\min_k \|f_\theta(x_i)[p] - \mu_{ip}\|^2$. This assigns pixel p to the cluster whose centroid minimizes the distance. Once all pixels have been assigned to clusters, the centroids $\mu_k$ are updated by taking the mean of the feature vectors of all the pixels assigned to each cluster. The centroid update rule is $\mu_k = \frac{1}{|C_k|}\sum_{(i,p)\in C_k} f_\theta(x_i)[p]$ where $C_k$ is the set of pixels assigned to cluster $k$. After each iteration, the assignments of pixels to clusters and the positions of the centroids are updated. The process converges when the assignments of pixels to clusters no longer change significantly, or after a predefined number of iterations.

The paper titled "Semantic Correspondence as an Optimal Transport Problem" introduces a novel approach for establishing dense correspondences between semantically similar images. The authors address two main challenges in semantic correspondence: the many-to-one matching problem and the background matching problem. Given feature maps $f_s \in \mathbb{R}^{h_s \times w_s \times d}$ and $f_t \in \mathbb{R}^{h_t \times w_s \times d}$, the standard approach is to compute a correlation map C between these features using cosine similarity:

$$C_{ijkl} = \frac{(f_s)_{ij} \cdot (f_t)_{kl}}{\left\|(f_s)_{ij}\right\| \|(f_t)_{kl}\|} \tag{5.19}$$

Where $C_{ijkl}$ represents the similarity score between the pixel at position $(i, j)$ in the source image and the pixel at position $(k, l)$ in the target image. The individual matching approach typically assigns the best match for each source pixel $(i, j)$ by maximizing $C_{ijkl}$ over $(k, l)$ :

$$(k^*, l^*) = \text{argmax}\, C_{ijkl} \tag{5.20}$$

However, this can result in many-to-one matching, where multiple source pixels match to the same target pixel, leading to suboptimal or ambiguous correspondences.

**PiCIE uses a non-parametric classifier:**: In a non-parametric classifier, there is no explicitly learned decision boundary or parametric model (like a neural network classifier). Instead, the classification is performed based on comparisons between data points and reference points (in this case, cluster centroids). The key idea is that the class of a pixel is determined by the similarity of its feature representation to the cluster centroids, without explicitly learning a classifier.

Once the centroids $\mu_k$ are computed and assigned to pixels, the non-parametric

classifier is responsible for assigning pixel labels during training. Instead of learning a classifier function (like a fully connected layer with weights) to predict the label of each pixel, PiCIE uses the distance between the pixel feature vector and the cluster centroids to compute a softmax probability that assigns the pixel to a cluster (label).

This is done by computing the cosine similarity between the pixel feature $f_\theta(x)[p]$ and each cluster centroid $\mu_k$, followed by a softmax function to convert these similarities into probabilities. The pixel is then assigned to the cluster with the highest probability. The classification loss is based on the softmax cross-entropy between the pixel feature and the cluster centroids:

$$L_{\text{clust}}\ \left(f_\theta\left(\vec{x}_i\right)[p], y_{ip}, \mu\right) = -\log\left(\frac{\exp\left(-d\left(f_\theta\left(\vec{x}_i\right)[p]\right), \mu_{y_{ip}}\right)}{\sum_l \exp\left(-d\left(f_\theta\left(\vec{x}_i\right)[p]\right), \mu_l\right)}\right) \tag{5.21}$$

Here, $\mathrm{d}(\cdot,\cdot)$ is the cosine distance between the feature and the centroid.

**Invariance to Photometric Transformations**: To make the clustering more robust, PiCIE introduces invariance to photometric transformations. Photometric transformations refer to changes in color, brightness, contrast, or other visual properties that do not alter the object structure. The goal is that the cluster assignments should not change under such transformations.

To enforce this, PiCIE generates two transformed versions of each image, $P^{(1)}\left(\vec{x}_i\right)$ and $P^{(2)}\left(\vec{x}_i\right)$ where P represents a random photometric transformation. For each pixel ppp, PiCIE computes the features under both transformations:

$$z_{ip}^1 = f_\theta\left(P^{(1)}\left(\vec{x}_i\right)\right)[p], z_{ip}^2 = f_\theta\left(P^{(2)}\left(\vec{x}_i\right)\right)[p] \tag{5.22}$$

PiCIE then performs k-means clustering on these two sets of feature vectors separately, resulting in two sets of cluster memberships and centroids, $(y^1, \mu^1)$ and $(y^2, \mu^2)$. The within-view loss encourages consistency within each view:

$$L_{\text{within}}\ = \sum_{i,p} L_{\text{clust}}\ \left(z_{ip}^1, y_{ip}^1, \mu^1\right) + L_{\text{clust}}\ \left(z_{ip}^2, y_{ip}^1, \mu^1\right) \tag{5.23}$$

The cross-view loss encourages the network to produce similar cluster assignments across different photometric transformations:

$$L_{\text{cross}}\ = \sum_{i,p} L_{\text{clust}}\ \left(z_{ip}^1, y_{ip}^2, \mu^2\right) + L_{\text{clust}}\ \left(z_{ip}^2, y_{ip}^1, \mu^1\right) \tag{5.24}$$

This forces the network to learn features that are robust to changes in photometric

properties.

**Equivariance to Geometric Transformations**: PiCIE also enforces equivariance to geometric transformations (e.g., scaling, cropping, rotation). Unlike photometric transformations, geometric transformations change the spatial arrangement of the image, and the labels should change accordingly. For geometric transformations, PiCIE applies a random geometric transformation $G_1$ (such as random cropping or flipping) to the image. The feature vectors are computed as follows:

$$z_{ip}^1 = f_\theta \left( G^{(1)} f_\theta \left( P^1 \left( \vec{x}_i \right) \right) \right) [p], z_{ip}^2 = G^{(2)} f_\theta \left( P^2 \left( \vec{x}_i \right) \right) [p] \tag{5.25}$$

The geometric transformation $G_1$ is applied both to the image and the feature maps. This ensures that the pixel labels in the transformed image correspond to the labels of the original image, up to the geometric transformation. The final loss is a combination of the within-view and cross-view losses:

$$L_{\text{total}} = L_{\text{within}} + L_{\text{cross}}$$

This ensures that the network learns features that are invariant to photometric changes but equivariant to geometric transformations.

**Overclustering:** To improve the stability of clustering, PiCIE also uses a technique called overclustering, where the number of clusters $K_1$ is larger than the actual number of semantic categories. Overclustering helps the network avoid collapsing multiple semantic categories into a single cluster. The final loss includes a balancing term:

$$L = \lambda_{k_1} L_{k1} + \lambda_{k_2} L_{k2} \tag{5.26}$$

Where $k_1$ and $k_2$ are the number of clusters, and $\lambda_{k_1}$ and $\lambda_{k_2}$ are balancing weights. The idea is to balance the clustering objective across different cluster granularities.

## 5.6 STEGO

The paper "Unsupervised Semantic Segmentation by Distilling Feature Correspondences" [18] proposes a method called STEGO (Self-supervised Transformer with Energy-based Graph Optimization). It is designed for unsupervised semantic segmentation. Given an image $I$, the algorithm extracts visual features using DINO such that $f = DINO(I) \in \mathbb{R}^{CxH}xW$. These features are mapped to a lightweight neural network S $\left( f \in \mathbb{R}^{KxH}xW \right)$ which map f into lowerdimensional embedding space where each feature corresponds to a particular semantic category. The embedding $S(f)$ is designed to

cluster features that belong to the same semantic class together. For example, pixels corresponding to the cat's face, body, and ears will be mapped to a similar region in the embedding space, indicating they all belong to the "cat" category. After the segmentation head maps features into the new code space, the algorithm applies a clustering step (like K-Means) to assign discrete semantic labels to the compact feature clusters. These clusters represent the final semantic segments of the image

**Detailed Methodology:** STEGO uses a pretrained self-supervised model (such as DINO) to extract dense feature maps from input images. The feature maps for two images x and y are denoted as $f(x) \in \mathbb{R}^{C \times H} \times W$ and $f(y) \in \mathbb{R}^{CxH}xW$ where C represents the number of channels, and H and W represent the spatial dimensions. For two images $x$ and $y$, STEGO computes the feature correspondence tensor F:

$$F_{hwij} = \sum_{c} \frac{f_{chw}}{|f_{hw}|} \frac{g_{cij}}{|g_{ij}|}$$

Here, $f_{chw} \in \mathbb{R}^{CxHxW}$ is the feature vector at pixel location $(h, w)$ of image x and $g_{cij} \in \mathbb{R}^{CXIXJ}$ is the feature vector at pixel location $(i, j)$ of image y.C is the channel dimension, and the spatial dimensions are represented by $(h, w)$ and $(i, j)$. The equation computes the cosine similarity between features in different spatial locations. This tensor helps capture the similarity between features at different spatial locations and is used to define correspondences between images.

STEGO learns a lightweight segmentation head S that projects the features $f(x)$ and $f(y)$ into a lower-dimensional space $s \in \mathbb{R}^{kxhxw}$ where K is smaller than the number of channels in the original feature space. The segmentation head's goal is to refine the feature correspondences and ensure they form compact clusters.

Let $s(x) = S(f(x))$ and $t(y) = S(f(y))$ be the projection outputs for images $x$ and $y$ respectively. The similarity between projected features is used to compute the segmentation correspondence tensor $S$ :

$$S_{hwij} = \sum_{c} \frac{s_{chw}}{\|s_{chw}\|} \cdot \frac{t_{cij}}{\|t_{cij}\|} \tag{5.27}$$

STEGO's central loss function compares the feature correspondence tensor $F$ and the segmentation correspondence tensor $S$ for the same pair of images. The objective is to align the segmentation features with the pretrained features. The simple correlation loss is defined as:

$$L_{\text{simple-corr}}(x, y, b) = \sum_{hwij} (F_{hwij} - b)\, S_{hwij} \tag{5.28}$$

Where b is a bias term added to prevent collapse by introducing "negative pressure". The aim is to maximize the agreement between F and S where the feature correspondences are strong, and to push them apart where the feature correspondences are weak.

**Feature Centering & Clamping:** To stabilize the optimization, the paper introduces two modifications:'
Spatial Centering: Adjusts the correspondence tensor F by centering it spatially to handle small objects better:

$$F^{SC}_{hwij} = F_{hwij} = \frac{1}{IJ} \sum_{i'j'} F_{hwi'j'}$$

Zero Clamping: Ensures that segmentation correspondences do not encourage total antialignment by clamping segmentation correspondences to zero when they are weakly correlated:

$$L_{corr}(x, y, b) = -\sum_{hwij} \left(F^{SC}_{hwij} - b\right) \max\left(S_{hwij}, 0\right)$$

These two modifications improve the stability of the loss function and ensure that smaller objects are better handled.

The correspondence distillation loss can be seen as minimizing an energy function over a graph of image pixels, akin to the Potts model:

STEGO further refines the learned correspondences by introducing KNNs. For each image, the algorithm finds its K-nearest neighbors in the feature space and computes the loss for each neighbor.

## 5.7 FDA

The intuitive idea behind the Fourier Domain Adaptation (FDA) paper [39] is centered around a simple but effective way to bridge the gap between two domains (e.g., synthetic images and realworld images) by focusing on the low-level image features that don't impact the high-level semantic content.

The core intuition of the paper is that low-frequency information in an image (such as global lighting, color, or texture) is often responsible for these domain shifts. However, high-frequency information (like edges, shapes, and object boundaries) contains

the essential information for understanding the image's semantic content (e.g., identifying a car, tree, or person).

So, the idea is:

- Low-frequency features (colors, lighting conditions) can be swapped between the source and target domains without affecting the high-level semantics (what objects are present in the image).
- By replacing the low-frequency content in the source image with the low-frequency content from the target image, we make the source image look more like a target image in appearance, while keeping the high-level content unchanged.

Fourier Transform: First, the source and target images are converted into the frequency domain using the Fourier transform. This breaks the images into low-frequency components (global features like color, brightness) and high-frequency components (fine details like edges and textures). The method swaps the low-frequency components of the source image with those from the target image, while keeping the high-frequency components (which contain the key semantic information) intact. This makes the source image "look" like a target image in terms of basic appearance (colors, lighting), but it keeps the important high-frequency information unchanged.

In the paper, the Fourier transform is applied to images to separate them into frequency components. For an image x of size $HxW$, the Fourier Transform is defined as:

$$F(x)(m,n) = \sum_{h,w} x(h,w) e^{-j2\pi\left(\frac{h}{H}m + \frac{w}{w}n\right)} \tag{5.29}$$

Here $F(x)(m,n)$ is the Fourier coefficient for the frequency pair and $e^{-j2\pi(,)}$ is the complex exponential. After the Fourier transform, the image is represented in terms of amplitude and phase components:

- Amplitude FA(x): The magnitude of the frequency component.
- Phase FP(x): The angle of the frequency component.

Mask $M_\beta$ to adjust the low-frequency components. This mask is defined around the lowfrequency region:

$$M_\beta(h,w) = 1_{(h,w)\in[\beta H;\beta H,-\beta W;\beta W]}$$

The parameter $\beta$ controls the size of the low-frequency region to swap. The core operation of FDA is to replace the low-frequency amplitude of the source image $x_s$ with that of the target image $x_t$ :

$$xs \to t = F^{-1}\left([M_\beta \circ FA(xt) + (1 - M_\beta) \circ FA(xs), FP(xs)]\right) \tag{5.30}$$

Here, $FA(xt)$ is the amplitude of the target image, and $FP(xs)$ is the phase of the source image. The inverse Fourier transform $F^{-1}$ converts the modified spectrum back to the image domain, creating an image $xs \to t$ that retains the structure of the source image but has the appearance of the target image.

After performing FDA, the transformed source images $xs \to t$ are used to train a segmentation network $\phi_w$ using a standard-cross entropy loss:

$$L_{ce}\left(\phi_w; D_{s\to t}\right) = -\sum_i \left\langle y_s^i, \log\left(\phi_w\left(xs \to t^i\right)\right)\right\rangle \tag{5.31}$$

This loss function encourages the network to predict the correct segmentation labels for the transformed source images.

The model also incorporates an entropy minimization loss on the target images to regularize the decision boundary:

$$L_{ent}\left(\phi_w; D_t\right) = \sum_i \rho\left(-\left\langle \phi_w\left(xt^i\right), \log\left(\phi_w\left(xt^i\right)\right)\right\rangle\right) \tag{5.32}$$

Here, $\rho(\mathrm{x}) = \left(x^2 + 0.001^2\right)\eta$ is a robust penalty function (Charbonnier loss) that penalizes highentropy (uncertain) predictions.

## 5.8 Semantic Correspondence

**Introduction to Semantic Correspondence:** Semantic correspondence refers to establishing dense, pixel-level associations between semantically related regions in two or more images. Unlike traditional image alignment techniques, which rely on geometric transformations, semantic correspondence focuses on identifying regions that share similar semantic content even if they differ in appearance, scale, or pose.

**Representing the Task:** Given two input images $I_1$ and $I_2$ of sizes $W_1 \times H_1 \times C$ and $W_2 \times H_2 \times C$, respectively, the goal is to compute a correspondence map $C(x_1, y_1) \to (x_2, y_2)$ that aligns each pixel $(x_1, y_1)$ in $I_1$ to a corresponding pixel $(x_2, y_2)$ in $I_2$ based on semantic similarity. This mapping is expressed as:

$$C(x_1, y_1) = \underset{(x_2, y_2)}{\operatorname{argmax}}\ \phi(F_1(x_1, y_1), F_2(x_2, y_2)) \tag{5.33}$$

where $F_1$ and $F_2$ are feature maps extracted from $I_1$ and $I_2$, respectively, and $\phi$ is a similarity function that measures semantic compatibility between features.

To understand the framework, suppose we have two $3 \times 3$ matrices $A$ and $B$, representing two images. Each element in these matrices corresponds to the intensity of a pixel. Semantic correspondence maps pixels from $A$ to $B$ based on a similarity metric or structural context.

**Matrix $A$: Original Image**

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

**Matrix $B$: Transformed Image**

$$B = \begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 8 \\ 1 & 4 & 7 \end{bmatrix}$$

Matrix $B$ is a rotated version of $A$ (90° clockwise). Semantic correspondence is to find which pixel in $A$ corresponds to each pixel in $B$. For semantic correspondence, the mapping can be represented as:

$$\text{From A (row, col)} \rightarrow \text{To B (row, col)}$$

$$(1,1) \rightarrow (1,3), \quad (1,2) \rightarrow (2,3), \quad (1,3) \rightarrow (3,3),$$

$$(2,1) \rightarrow (1,2), \quad (2,2) \rightarrow (2,2), \quad (2,3) \rightarrow (3,2),$$

$$(3,1) \rightarrow (1,1), \quad (3,2) \rightarrow (2,1), \quad (3,3) \rightarrow (3,1).$$

A simple example of correspondence is given two images of the same car taken from different angles, find the car's headlights in one image to the headlights in the other. Even though the positions or orientations of headlights differ, a correct algorithm should be able to align features based on meaning and not just appearance or location. Thus, the losses employed in correspondence tasks should ensure accurate matching of semantically similar regions while preserving structural and geometric properties across images.

Often, the typical losses model the shift from source to target image as a transfor-

mation. Given this, **correspondence loss** is defined as thus,

$$\mathcal{L}_{\text{correspondence}} = \frac{1}{N}\sum_{i=1}^{N} \|T_s(x_i) - x_i'\|_2^2 \tag{5.34}$$

Where $T_s$ is transformation applied to source points. Another natural choice is the **Cycle-Consistency Loss** which enforces that transformations applied forward and backward map points back to their original positions, ensuring consistency:

$$\mathcal{L}_{\text{cycle}} = \frac{1}{N}\sum_{i=1}^{N} \|T_t(T_s(x_i)) - x_i\|_2^2$$

Where $T_s$ is Source-to-target transformation and $T_t$ is Target-to-source transformation. This ensures the mappings are bijective and robust. Since correspondence requires a braoder understanding of meaning, **semantic alignment** loss often aligns high-dimensional feature embeddings of semantically similar regions between images:

$$\mathcal{L}_{\text{alignment}} = \frac{1}{N}\sum_{i=1}^{N} \|\phi_s(x_i) - \phi_t(x_i')\|_2^2 \tag{5.35}$$

Where $\phi_s$ is the Feature embedding function for the source image. and $\phi_t$ is the Feature embedding function for the target image.

This ensures semantically similar regions have consistent feature representations.

## 5.9 Unsupervised Semantic Correspondence Using Stable Diffusion

Let us consider the paper, "Unsupervised Semantic Correspondence Using Stable Diffusion" [19]. The main point of the paper is that pre-trained diffusion models specifically Stable Diffusion can be used for tasks beyond image generation-specifically for finding semantic correspondences between images, without any additional training or supervision.

Firstly, let us understand how we can define **local semantic correspondence using attention maps**. Let $\mathcal{J}_s$ and $\mathcal{J}_t$ represent the source image and target image, respectively. Let $p_s \in \mathcal{J}_s$ be the query point in the source image. We aim to find areas in the target image $\mathcal{J}_t$ that semantically correspond to $p_s$. First, we extract feature representations from both images. Assume we have a feature extractor $f$ (.) (e.g., a CNN or Transformer backbone) that maps each pixel or patch in the image to a high-dimensional feature space:

$$F_s = f\left(\mathcal{J}_S\right), F_t = f\left(\mathcal{J}_t\right) \tag{5.36}$$

Where $F_s$ and $F_t$ are feature maps corresponding to the source and target images. Each feature map represents the features at various spatial locations in the respective images.

Let $f\left(p_s\right) \in \mathbb{R}^d$ be the feature vector corresponding to the query point $p_s$ in the source image, where d is the dimension of the feature space. We now compute an attention map to highlight areas in the target image that correspond to the query point $p_s$. To compute this, we calculate a similarity score between the feature vector $f\left(p_s\right)$ and the feature vectors at every position in the target image feature map $F_t$. This could be done using a dot product or cosine similarity. For simplicity, we use the dot product here:

$$\operatorname{sim}\left(f_s\left(p_s\right), f_t\left(J_t\right)\right) \tag{5.37}$$

This similarity score measures how aligned or "similar" the feature representations of the query point in the source image and the points in the target image are. To create an attention map, we normalize the similarity scores using a softmax function over all positions in the target image:

The final attention map is a 2 D grid of attention weights $\alpha\left(p_t\right)$ which highlights the regions in the target image that are most semantically similar to the query $p_s$ in the source image. In essence:

$$A_t\left(p_t\right) = \left\{A_t\left(f_s\left(p_s\right), f_t\left(\mathcal{J}_t\right)\right) \mid p_t \in I_t\right\} \tag{5.38}$$

The authors in the paper use stable diffusion model II as their feature extractors $f(.)$. In particular, they utilize the cross-attention maps $A_{\text{cross}}$ . Given a source image $\mathcal{J}^S$ and a query point $u$ in the image $\mathcal{J}^S$ (such as a paw), they optimize an embedding $e$ so that the attention map $A_t$ of that embedding $e$ with the source image $\mathcal{J}^s$ highlights the query location in the source image.

During inference, this optimized embedding $e$ creates a cross-attention matrix map $A_\alpha$ with the target image $\mathcal{J}^t$. They take the maximum of this attention map $A_\alpha$ to find the location $u_p$ that points to a region semantically similar to $u$ (see the figure below)

**Method:** Given an image $I$, we use a VQ-VAE to map it to its encoder representation $\vec{z}_0(t)$. Subsequently, we noise it using the forward process $\vec{z}_T(t)$. We then perform

denoising using the U-NeT to map $\vec{z}_t(t)$ to $\vec{z}_{t-1}(t)$ progressively until we obtain the latent $\vec{z}_0(t)$. This conditioning is performed using a text-embedding $\vec{e}$ which is randomly initialized. In particular, if you look at the code, it is defined inside "optimize_prompt" with the variable context $= \vec{e}$ being the vector that is optimized. The cross-attention at the $l-th$ layer of the U-Net within the diffusion model is defined as:

$$M_l(\vec{e}, l) = \text{softmax}\left(\frac{Q_l K_l^T}{\sqrt{d_l}}\right) \tag{5.39}$$

Where, $Q_l = \phi_l(\vec{z}(t)) \in \mathbb{R}^{Cx(h\times w)xd_l}$ is the query obtained from the image features $\vec{z}(t)$, $K_l = \psi_l(\vec{e}) \in \mathbb{R}^{CxPxd_l}$ is the key obtained from the text embedding $\vec{e}$, $d_l$ is the dimension of the latent space at layer $l$ and $M_l(e,l) \in \mathbb{R}^{Cx(hxw)xP}$ is the attention map, with h, w representing spatial dimensions, P is the number of textual tokens and C the number of attention heads.

To reiterate, the attention map $M_l(e,l)$ assumes the following form $M_l(e,l) \in \mathbb{R}^{Cx(hxw)xP}$ where h x w are the spatial dimensions, P is the number of textual tokens and is the number of heads

The first thing that authors do is that they average across the Channels or number of heads. As a result, the attention heads become:

$$M_c(e,l) = \text{avg}_{\text{channel}}\ (M_l(e,l)) \in \mathbb{R}^{(hxw)xP} \tag{5.40}$$

To take advantage of the different characteristics of each layer, the authors average along both the channel axis and across a subset of U -Net layers $M_c(e,l) \in \mathbb{R}^{(hxw)xP}$. Obviously, the size of attention maps is different, so what they do is they perform bilinear interpolation when averaging. Thus, they obtain:

$$M_{u\_net} = avg_{\text{unet}}\ (M_c(e,l)) \in \mathbb{R}^{(hxw)xP} \tag{5.41}$$

In the code, the function responsible for performing bilinear interpolation is `upscale_to_image_size`. When handling an attention map comprising 1024 tokens, corresponding to a $32 \times 32$ grid, this function first reshapes the 1024 tokens into a $32 \times 32$ grid. Subsequently, it applies bilinear interpolation to upscale the $32 \times 32$ grid to a $512 \times 512$ resolution.

Finally, the authors pick an attention map associated with the first text token p. In particular,

$$M''(u_s, I, e) = M_{u\_net}[1] \in \mathbb{R}^{(hxw)} \tag{5.42}$$

Where the authors do not employ the first or last token as typically these are special termination token. They also state that they empirically observed that the choice of

which token does not have a significant impact on the final outcome as all other tokens (i.e., $P$ entries of $\vec{e}$ ) all are also optimized regardless due to the softmax that we apply along the P axis - optimization will find the prompts (or more exactly the embeddings) that match the chosen token location.

**Optimization:** For a given source image $I_s$ and a query location $u_s \in [0,1]^2$, the authors aim to optimize a text embedding e such that the attention map $M(u; e, I_s)$ focuses on the region of interested centered around $u_s$. To understand this, imagine you have two images:

1. Source image $I_i$ : An image of a cat.
2. Target Image $I_j$ : An image of a dog.

We want to find a pixel in the dog image $\boldsymbol{I}_j$ that corresponds to a specific pixel (say, the pixel that represents the cat's ear) in the cat image $\boldsymbol{I}_i$. The task is to find the pixel in $\boldsymbol{I}_j$ that represents the dog's ear, which is semantically similar to the cat's ear in the source image.

Let us say we have a query pixel $u_i$ in the cat image that corresponds to the cat's ear. The pixel location is normalized to the image size, meaning it points to a specific pixel in the source image (e.g., normalized coordinates might be $\boldsymbol{u}_i = (0.3, 0.5)$ which corresponds to the center of the cat's ear). The task is to find the semantically corresponding pixel $u_j$ in the dog image $I_j$ which should point to the dog's ear.

Now, we already have $M''(u_s, I, e)$. The problem with this map is that it might be too spread out or not focused on the specific region we care about because $e$ was initialized randomly. The model's attention could be distributed across many irrelevant areas in the image. For example, if we are trying to focus on the dog's ear, the initial attention map might focus on multiple regions (like the dog's tail, legs, etc.), because the model hasn't yet learned to concentrate its attention on the specific query pixel (remember, we used a random query e for now).

We must now learn to make the attention map of $e$ which is $M''(u_s, I, e)$ to focus on the region of our interest. In order to do this, we define our region of interest $u_s$ in the source image. At this query location $u_s$, we define a Gaussian of standard deviation $\sigma$ centered at $u_s$

$$M_S(u) = \exp\left(-\frac{\|u - u_S\|_2^2}{2\sigma^2}\right) \tag{5.43}$$

$M_s(u)$ is calculated for every pixel u in the image, representing the attention assigned to each pixel based on its distance from the query pixel $u_i$. Thus, given that the

query pixel is dog's ear, centered at $(100, 150)$, we would define $M_s(u)$ over the entire ( $512 \times 512$ grid). The optimization objective is to find the text embedding $\vec{e}^*$ that minimizes the difference between the model's unfocused attention map $M''(u_s, I, e)$ and the Gaussian map that focuses on our region of interest $M_S(u)$ :

$$\vec{e}^* = \text{argmin}_{\vec{e}} \sum_u \|M''(u; e, I_s) - M_s(u)\|_2^2 \tag{5.44}$$

Assume the dog's ear is located at pixel $u_s = (100, 150)$ in the image. We want the model to focus its attention primarily on this pixel and nearby pixels. To do this, we define a Gaussian distribution centered at the pixel $u_i$ : $M_i(u) = \exp\left(-\frac{\|u-u_i\|_2^2}{2\sigma^2}\right)$. Here, $u$ represents all the pixel locations in the image (for instance, all possible u = (x, y) pixel pairs). The Gaussian map $M_i(u)$
will have the highest value at the dog's ear and gradually taper off as you move away from the ear. If we set $\sigma = 10$, the Gaussian distribution will have a tight focus around the dog's ear.

The optimization process in equation (6) adjusts the text embedding $e$ to minimize the difference between the attention map $M''(u; e, I_s)$ and $M_s(u)$.

Is the MSE loss enough to capture the fine-grained patterns of the attention map at the query $q$ ?

Once the optimized text embedding $\vec{e}^*$ is obtained, it can be applied to a target image $I_t$ to find the semantically corresponding region. The corresponding pixel $u_t$ in the target image is determined by finding the maximum attention value:

$$u_t = \text{argmax}_u M(u; e^*, I_t) \tag{5.45}$$

This step involves computing the attention map for the target image and identifying the pixel with the highest attention score, which corresponds to the region that is semantically similar to the query region in the source image.

**Optimization across transformations:** The optimization of text embeddings on a single image is prone to overfitting. To address this issue, the authors propose averaging across image crops. For example, when optimizing the attention map for a specific region, such as the dog's ear in the source image, the training process involves presenting cropped versions of the source image, where only 93% of the image is visible. For each crop, the model generates an attention map, repositions it back into the full image's coordinates, and compares it to a Gaussian map that is similarly cropped and repositioned. The cropping is applied randomly across the image, ensuring that other regions maintain their relative positions.

The model adjusts its parameters to align the attention map with the Gaussian map across various crops, allowing it to focus on the target region, such as the dog's ear, regardless of cropping. During inference, a similar averaging strategy is employed to predict the corresponding pixel $u_j$ in the target image $I_t$. For a given crop $\mathcal{C}$, the operation $\mathcal{U}$ places the attention map for the cropped image back into the full image's coordinate system. Attention maps are generated for multiple cropped views of the target image, repositioned, and averaged to produce a robust attention map. The final step involves taking the argmax over the pixel locations $u$ to determine $u_j$, the pixel in the target image with the highest attention, corresponding to the semantic match for the query pixel in the source image.

**Averaging Across Crops** Let $\mathcal{C}(I)$ represent the cropping operation with parameters $c$, and $\mathcal{U}_c(x_c)$ denote the placement of the crop back to its original location, satisfying $\mathcal{C}_c(\mathcal{U}_c(x_c)) = x_c$ for some crop $x_c$. The image dimensions are reduced to 93% (determined via hyperparameter tuning), and a uniformly random translation is applied, denoted as $c \sim D$. The optimization process in Equation (6) is augmented by averaging across cropping augmentations as:

$$\vec{e}^* = \text{argmax}_{\vec{e}} \mathbb{E}_{c\sim D} \sum_u \left\| \mathcal{C}_c \left( M''(u; e, I_s) \right) - \mathcal{C}_c \left( M_s(u) \right) \right\|_2^2 \tag{5.46}$$

Similarly, during inference, attention masks from different crops are averaged:

$$u_j = \text{argmax}_u \, \mathbb{E}_{c\sim D} u_c \left( M(u; e^*, I_t) \right) \tag{5.47}$$

**Averaging Across Optimization Rounds** Empirical results show that performing multiple rounds of optimization is crucial for achieving strong performance, as illustrated in the figure below.

The optimization process in Equation (8) can be abstracted as $\vec{e}^* = \mathcal{O}(\bar{e}, I_i)$, where $\bar{e}$ represents the initialization. The attention masks obtained from multiple optimization runs are then averaged to improve robustness:

$$u_j = \text{argmax}_u \, \mathbb{E}_{e\sim D} M(u; \mathcal{O}(\bar{e}, I_i), I_J) \tag{5.48}$$

Each optimization round begins with a different initialization of the embedding $\vec{e}_j$, and the results are aggregated to produce a more reliable attention map. This approach ensures that the attention maps are robust to variations in initialization and cropping, resulting in better alignment with the target regions.

## 5.10 Unsupervised Keypoints

The paper, "Unsupervised Keypoints from Pretrained Diffusion Models," [19] build on the previous work of utilizing the already-encoded semantic knowledge encoded in Diffusion Models for downstream tasks. The main objective is to locate "keypoints" in images—distinct and semantically important regions—without relying on annotated data.

These Keypoints are essential in computer vision tasks like 3D reconstruction and motion tracking. These points serve as essential features that represent the geometry and structure of objects. One of their key advantages is transformation invariance, meaning they remain stable under scaling, rotation, and translation. This stability allows keypoints to generalize well, helping algorithms recognize objects regardless of context or perspective.

Keypoints can be classified into two types: supervised and unsupervised. Supervised keypoints are determined using annotated data where humans specify important landmarks, such as the corners of the mouth or the position of the eyes in facial recognition tasks. In contrast, unsupervised keypoints are automatically discovered without labeled data, relying on models to learn patterns or properties directly from images.

An example of keypoints' application is in human pose estimation, such as tracking a person in a video for activity recognition (e.g., Tai Chi movements). In a supervised approach, the method would rely on annotated datasets where humans have labeled joints like the head, shoulders, elbows, and knees in thousands of frames. These labeled landmarks serve as the ground truth for training. On the other hand, the unsupervised approach discussed in this paper avoids the need for annotations. Instead, the model identifies consistent patterns across multiple images of people performing Tai Chi.

The method relies on the cross-attention layers within diffusion models, where each text embedding token $\vec{e}$ connects to specific areas in the image, captured in an attention map $M(\vec{e}, X)$. The attention map is calculated as:

$$M(\vec{e}, \mathrm{X}) = E_C \left[ \mathrm{softmax}_n \left( \frac{Q_L^c \cdot K_l^c}{\sqrt{D_l}} \right) \right] \tag{5.49}$$

Where $E_c$ denotes averaging over transformer heads c.

The model begins with attention maps $M(\vec{e}, X)$ that are dispersed and exhibit activity spread across the entire image. To ensure the attention maps focus on specific, compact regions, the method introduces a localization mechanism. This mechanism works by defining a Gaussian distribution $G_n$ centered at the peak location of the attention map $M_n$ for each token $n$.

The process starts by identifying the peak location of $M_n$, which corresponds to the

pixel with the highest response in the attention map. This peak, $\mu_n$, is mathematically calculated as:

$$\mu_n = \text{argmax}_{w,h} M_n[h, w] \tag{5.50}$$

Using this peak, a Gaussian $G_n$ is created, centered at $\mu_n$, and is expressed as:

$$G_n = \exp\left(-\frac{\|XY_{\text{coord}} - \mu_n\|^2}{2\sigma^2}\right) \tag{5.51}$$

Here, $XY_{\text{coord}}$ represents the spatial coordinates of the image and $\sigma$ controls the spread of the Gaussian distribution. The Gaussian $G_n$ serves as the ideal localized representation that the attention map $M_n$ should emulate.

To encourage this localization behavior, a localization loss $\mathcal{L}_{\text{localize}}$ is defined, which measures the difference between the attention map $M_n$ and the target Gaussian $G_n$. This loss is calculated as:

$$\mathcal{L}_{\text{localize}} = \mathbb{E}_n \|M_n - G_n\|^2 \tag{5.52}$$

This loss compels the attention maps to become compact and focused, aligning closely with the Gaussian distributions centered at their respective peaks. In addition to localization, the method enforces consistency of attention maps under image transformations. This is achieved through the equivariance loss $\mathcal{L}_{\text{equiv}}$, which ensures that the attention maps remain consistent when the input images undergo small transformations $T$, such as rotations, translations, and scaling. The equivariance loss is expressed as:

$$\mathcal{L}_{\text{equiv}} = \mathbb{E}_n \|T^{-1}(M_e(e, T(X))) - M_n(e, X)\|^2 \tag{5.53}$$

This loss ensures that the model maintains a stable and reliable representation of keypoints, regardless of minor geometric changes in the input images.

The overall training objective combines these two losses into a single objective function:

$$L_{\text{total}} = \mathcal{L}_{\text{localize}} + \lambda_{\text{equiv}} \mathcal{L}_{\text{equiv}} \tag{5.54}$$

Here, $\lambda_{\text{equiv}}$ is a hyperparameter that balances the contribution of the equivariance loss relative to the localization loss. This combined objective ensures that the attention maps are both localized and robust to transformations, enabling the model to identify semantically meaningful and consistent keypoints across diverse images. This

methodology allows the model to generalize effectively, even when the input data varies significantly in pose, scale, or background conditions.

## 5.11 Self-Attention Guidance

**Self-attention maps** play a critical role in semantic segmentation within diffusion models by enabling fine-grained feature representation. These maps capture the pairwise relationships between every pixel in an image, allowing the model to learn how each region of the image influences others. This is particularly essential in semantic segmentation where understanding the global context and dependencies between spatial regions is necessary for accurate delineation of object boundaries and classes. Keeping this in mind, the work, "Improving Sample Quality of Diffusion Models Using Self-Attention Guidance" [22] is of special interest since it utilizes self-attention maps for downstream tasks.

Recall that there are two types of guidance: **Classifier Guidance (CG)** and **Classifier-Free Guidance (CFG)**. CG uses a trained classifier $p(c|\mathbf{x}_t)$ to guide the reverse process toward specific class distributions. The noise prediction is modified as follows:

$$\tilde{\epsilon}(\mathbf{x}_t, c) = \epsilon_\theta(\mathbf{x}_t) - s\sigma_t \nabla_{\mathbf{x}_t} \log p(c|\mathbf{x}_t),$$

where $s$ is the guidance scale controlling the strength of the classifier's influence.

CFG simplifies the guidance process by interpolating between conditional and unconditional predictions, formulated as:

$$\tilde{\epsilon}(\mathbf{x}_t, c) = \epsilon_\theta(\mathbf{x}_t) + (1+s)(\epsilon_\theta(\mathbf{x}_t, c) - \epsilon_\theta(\mathbf{x}_t)).$$

CFG does not require a separate classifier but relies on external labels or prompts, making it inapplicable to fully unconditional models. The proposed Self-Attention Guidance (SAG) method utilizes internal self-attention maps to guide the diffusion process. Unlike CG or CFG, SAG is condition-free and relies on internal representations. The guidance noise is computed as:

$$\tilde{\epsilon}(\mathbf{x}_t) = \epsilon_\theta(\mathbf{x}_t) + (1+s)(\epsilon_\theta(\mathbf{x}_t) - \epsilon_\theta(\widehat{\mathbf{x}}_t)), \tag{5.55}$$

where $\widehat{\mathbf{x}}_t$ is the selectively blurred version of $\mathbf{x}_t$, determined by the self-attention map $A_t$.

The aggregated self-attention map is obtained via global average pooling (GAP):

$$A_t = \text{Upsample}(\text{Reshape}(\text{GAP}(A_t^S))), \tag{5.56}$$

where $A_t^S$ denotes the stacked self-attention maps across heads.

To understand how selective self-guidance is employed in 5.55, consider the self-attention map $A_t$ at each timestep t . The attention map encodes information about which regions of the image or feature map are most important for the model's current prediction. Once the attention map $A_t$ is computed, the next step is to decide which regions are important enough to be blurred. This is done by applying a threshold $\psi$ to the attention map. Specifically, the regions that have attention scores above the threshold are considered "important" and are the ones that will be blurred. Mathematically, this is represented as:

$$M_t = 1\left(A_t > \psi\right) \tag{5.57}$$

Where $M_t$ is a binary mask created from the attention map, and 1 is the indicator function that assigns a value of 1 to regions where $A_t > \psi$ and 0 otherwise. After creating the mask $M_t$ the regions identified by the mask where $M_t = 1$ are blurred. The idea here is to blur only the regions that the model attends to the most-those areas that the model considers important for image generation. By doing this, the model is forced to refine these regions further during the denoising process.

The blurred version $\tilde{x}_t$ of the image is combined with the original image $x_t$ using the mask $M_t$ :

$$\tilde{x}_t = (1 - M_t) \odot x_t + M_t \odot \tilde{x}_t \tag{5.58}$$

Blurring the most important regions (as identified by the attention map) creates an adversarial scenario. The model is effectively challenged to restore the fine details in the blurred regions during the denoising process. Since these regions are crucial for image generation, the model is forced to focus more on refining and improving them, leading to higher-quality output. This is an important aspect of SAG: it forces the model to pay more attention to important areas by intentionally blurring them and then requiring the model to restore these regions during the diffusion process.

Empirical results show that SAG improves sample quality across various models which include ADM and Stable Diffusion. Metrics such as FID and IS indicate consistent improvements. Moreover, SAG is shown to be orthogonal to CG and CFG and complements them.

## 5.12 Label-Efficient Segmentation

The paper "Label-Efficient Semantic Segmentation with Diffusion Models" [5] shows how Diffusion Probabilistic models (DDPMs) can extract meaningful pixel-level representations for image segmentation in low-data settings.

In this paper, **representation learning** is applied by leveraging the intermediate activations of DDPMs during the reverse diffusion process as pixel-wise semantic features for segmentation tasks. Thus, it shows that these features are *generalizable* to the task of segmentation even though DDPMs were originally trained for generative purposes.

For a noised latent $\vec{z}_T$, the diffusion model's noise prediction network (often parameterized by a U-Net) produces a series of feature maps $f_1, f_2, ..., f_n$ across the different layers of the U-Net architecture at each timestep t . Let $f_\theta\left(\vec{z}_t, t\right)$ be the feature map produced by the U-Net decoder at timestep $t$ and $B_i$ denote the feature map from the i-th block of the U-Net decoder. The feature map at block $B_i$ will have a spatial resolution $H_i \times W_i$ which is typically smaller than the input resolution $HxW$. The feature maps are upsampled to the original image resolution $HxW$ using bilinear interpolation to form pixel-level representations:

$$\hat{f}_\theta\left(\vec{z}_i, B_i\right) \in \mathbb{R}^{H \times W \times C_i} \tag{5.59}$$

Where $C_i$ is the number of channels in the feature map. For segmentation, the authors extract representations from the middle layers of the U-Net decoder across several reverse diffusion steps:

$$\left\{B_5, B_6, B_7, B_8, B_{12}\right\}, t \in \{50, 150, 250\} \tag{5.60}$$

These timesteps are chosen together with the middle-blocks of the U-NeT for features because empirically the authors find that they tend to capture the most useful semantic information for segmentation.

The feature maps from these layers are concatenated to form a high-dimensional pixel-wise feature vector $\vec{f_p} \in \mathbb{R}^{HxWxD}$ where $D = \sum_i C_i$ is the total number of channels

Once the pixel-level representations are extracted, the goal is to predict a semantic label for each pixel. Since the method operates in a few-shot learning regime, only a small number of labeled images are available. For each pixel $p$, the concatenated feature vector $\vec{f_p}$ is passed through a multi-layer perceptron (MLP) classifier, which predicts the pixel's class label:

$$\hat{y}_i = MLP\left(\vec{f}_p\right) \tag{5.61}$$

Where $\vec{y}_p = \{1, 2, \ldots K\}$ represents the predicted class label for pixel p and K is the total number of semantic classes. The MLP consists of two hidden layers with ReLU activations and batch normalization, followed by a softmax output layer to predict the class probabilities.

The MLP classifiers are trained using the pixel-wise representations extracted from the few labeled images. The loss function used to train the MLP is the cross-entropy loss between the predicted and ground-truth pixel labels

$$\mathcal{L}_{\text{seg}} = \sum_{i=1}^{N} \text{CE}(y_i, \hat{y}_i) \tag{5.62}$$

where $y_i$ is the ground truth, $\hat{y}_i$ is the predicted label, and CE is the cross-entropy loss.

For a test image, we pass it through the DDPM and extract pixel-wise features. We then use the trained MLP to predict a class for each pixel. Using an ensemble of MLPs, we predict the final label using majority voting.

## 5.13 Diff-Attend Segment

**Non-Maximum Suppression:** Assume we have a set of $N$ candidate regions $\{R_1, R_2, \ldots, R_N\}$. Each region $R_i$ has:

1. **A confidence score** $S_i \in [0, 1]$ which measures how likely it is to contain the target object.
2. **Bounding box coordinates** $B_i$ often denoted as $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ for each box $i$.

The first step in NMS is to sort the regions by their confidence scores in descending order:

$$\{R_1, R_2, \ldots, R_n\} \quad \text{s.t.} \quad S_1 \geq S_2 \geq \cdots \geq S_N.$$

The Intersection over Union (IoU) measures the overlap between two bounding boxes, defined as:

$$\text{IoU}\left(B_i, B_j\right) = \frac{\text{Area}\left(B_i \cap B_j\right)}{\text{Area}\left(B_i \cup B_j\right)}.$$

Using an IoU threshold $T \in [0, 1]$, we proceed as follows:

1. Initialize an empty set $M$ to store the selected regions.
2. Iterate through each region in descending order of confidence score:
   (a) Select the current highest scoring region $R_i$ and add it to $M$.
   (b) For each remaining region $R_j$, calculate the IoU between $R_i$ and $R_j$:

$$\text{IoU}\left(B_i, B_j\right).$$

3. Suppress $R_j$ if:

$$\text{IoU}\left(B_i, B_j\right) > T.$$

Continue this process until all regions have either been selected or suppressed. At the end, $M$ contains the non-suppressed, highest-confidence regions.

**Example:** Let's do a simple example of NMS with 3 hypothetical regions $R_1, R_2$, and $R_3$, with confidence scores $S_1 = 0.9$, $S_2 = 0.8$, and $S_3 = 0.75$. Suppose the IoU threshold $T = 0.5$. Sort the regions by confidence scores: $S_1 = 0.9$, $S_2 = 0.8$, and $S_3 = 0.75$.
Start with $R_1$ (the region with the highest confidence score) and compare it with $R_2$ and $R_3$ using the IoU formula. Assume:

The paper, titled "Diffuse, Attend, and Segment: Unsupervised Zero-Shot Segmentation using Stable Diffusion," [34] introduces DiffSeg, an innovative method for unsupervised, zero-shot image segmentation. This approach leverages the self-attention layers of stable diffusion models to generate high-quality segmentation masks. The primary objective of DiffSeg is to enable the segmentation of any image without the need for pre-existing labels or supervision, utilizing the inherent ability of the self-attention layers to capture semantic groupings within their attention tensor

In the U-Net, each self-attention layer produces 4-dimensional tensors, which contain the spatial attention map for different resolutions. These attention tensors are denoted as:

$$\mathrm{A} \in \left\{ A_k \in \mathbb{R}^{h_k \times w_k \times h_k \times w_k} \mid k = 1, \ldots, 16 \right\} \tag{5.63}$$

where each tensor $A_k$ represents attention values at a particular resolution, with each dimension corresponding to spatial locations within that layer.

The authors introduce two properties of the attention maps which are essential for segmentation:

- **Intra-Attention Similarity**: Within a single 2D attention map $A_k[i, j, :, :]$ (for any given pixel (i,j)), pixels that belong to the same object tend to have high similarity in their attention values.
- **Inter-Attention Similarity**: Between different 2D attention maps, such as $A_k[i, j, :, :]$ and $A_k[i + 1, j + 1, :, :]$ similar attention values indicate that these areas likely belong to the same object across locations.

These properties suggest that self-attention layers inherently group image regions that share similar features, making them ideal candidates for segmentation. Each attention map, $A_k$, represents a different spatial resolution. To create a unified high-resolution representation, the attention maps are aggregated:

1. The attention maps are upsampled to the highest resolution, $64 \times 64$, using bilinear interpolation:

$$\tilde{A}_k = \text{ Bilinear-upsample } (A_k) \in \mathbb{R}^{h_k} \times w_k \times 64 \times 64 \tag{5.64}$$

2. Each upsampled attention map $\tilde{A}_k$ is assigned a weight $R_k$ proportional to its resolution, emphasizing high-resolution maps for detailed segmentation.

The final aggregated tensor, $A_f$, is calculated by summing these weighted attention maps:

$$A_f[i,j,:,:] = \sum_{k=1}^{16} \tilde{A}_k \left[ \frac{i}{\delta_k}, \frac{j}{\delta_k}, :, : \right] \cdot R_k \tag{5.65}$$

Where $\delta = \frac{64}{w_k}$ controls the scaling for each resolution level, ensuring all attention maps are spatially aligned.

**Iterative Attention Merging:** This step leverages KL divergence to identify and merge regions of high similarity across the attention map:

1. Anchor Points: A grid of anchor points, M × M, is generated from the aggregated attention tensor $A_f$. Each anchor point has an associated attention map $\mathcal{L}_a$, serving as a segmentation "seed": $\mathcal{L}_a = \{A_f\,[i_m, j_m, :, :] \in \mathbb{R}^{64\times64} \mid (i_m, j_m) \in M\}$

2. Merging Criterion: The similarity between any two attention maps is measured using a symmetric KL divergence:

$$D\left(A_f[i,j], A_f[y,z]\right) = KL\left(A_f[i,j] \| A_f[y,z]\right) + KL\left(A_f[y,z] \| A_f[i,j]\right) \tag{5.66}$$

3. Iterative Merging: In each iteration, pairs of attention maps with similarity $D < \tau$ (a predefined threshold) are merged by averaging them. This iterative process gradually combines similar regions into unified object proposals, creating a list of object masks, $\mathcal{L}_p$

**Non-Maximum Suppression (NMS):** After merging, the remaining maps are refined to create the final segmentation mask. Each location in the image is assigned to the attention map with the highest probability, ensuring only the most likely segmentation label remains for each pixel. The resulting segmentation mask, $S \in \mathbb{R}^{64\times64}$ is produced by subsampling $\mathcal{L}_p$ and then applying NMS:

$$S[i,j] = \arg\max \mathcal{L}_p[:,i,j] \tag{5.67}$$

To understand the framework, consider an example where we have an input image and we want to segment two main objects, say a cat and a dog. The image is processed by the U-Net component in Stable Diffusion. The U-Net generates 16 self-attention

maps $A_k$ ( for k = $1, 2, \ldots 16$ ) each representing a different spatial resolution. Attention maps come in four resolutions ( 8 x 8 ), ( 16 × 16 ), ( 32× 32) and (64 x 64). DiffSeg aggregates these self-attention maps into a single high-resolution map $64 \times 64$ to combine information from all layers: $\tilde{A}_k =$ Bilinear Upsample $(A_k)$. We then assign higher weights to maps with finer details (higher resolution maps), and combine them into a single high-resolution attention tensor: $A_f[i, j, : \ . \ :] = \sum_{k=1}^{16} \tilde{A}_k[i, j, :, :] \times R_k$. Here $R_k$ is a weight that reflects the importance of each map. This results in an aggregated attention tensor $A_f$ of size $64 \times 64$. To make $A_f$ a probability distribution, normalize it such that each slice $A_f[i, j, : \ . \ :]$ sums to 1 . We then divide $A_f$ into anchor points by sampling locations on a grid. Let's say we create a $16 \times 16$ gride of anchors, so each anchor corresponds to a region within $A_f$. For each region $A_f[i, j, : \ . \ :]$ calculate the KL divergence with neighboring anchors. This measures how similar two attention maps are, indicating whether they belong to the same object. Start with the highest-scoring region and merge it with others that have a KL divergence below a threshold $\tau$. Continue merging until no more regions with a similarity below $\tau$ remains. This process iterates N times, gradually reducing the number of regions and forming coherent object groups. After iterative merging, we have a set of object proposals in the form of attention maps. NMS is used to assign each pixel to only one of these proposals. Each object proposal $L_p$ is unsampled to match the original image resolution. For each pixel $(i, j)$ look at the attention values across all object proposals $L_p[k, i, j]$. Assign the pixel to the proposal with the highest confidence:

$$S[i, j] = \arg\max_k L_p[k, i, j] \tag{5.68}$$

This ensures that each pixel belongs to only one object in the final segmentation map S. After applying NMS, S is a clean segmentation mask where each pixel is assigned to the most confident proposal, creating clear boundaries around each object (e.g., cat and dog).

## 5.14 DiffewS

DiffewS [41] performs **few-shot image segmentation**. Imagine a scenario where a wildlife researcher has only a few annotated images of a rare species of bird, and they want to automatically segment images of this bird in a large collection of unlabeled photos. In this case, the Few-shot Semantic Segmentation (FSS) model in the paper would take a few labeled examples (support images of the bird with a mask highlighting the bird) and use them to guide segmentation on new, unlabeled images (query images).

DiffewS establishes a relationship between the bird's features in the support and

query images. It modifies the attention mechanism to let the model focus on similar features across both images, helping it find the bird in new contexts or backgrounds. The support images have labeled masks showing the bird which helps the model understand what parts of the image are important. For instance it can focus on the bird's color, shape, and texture in the support images then apply this knowledge to detect similar regions in the query images.

The core of DiffewS is the **KV Fusion Self-Attention** which combines the key-value pairs from both support and query images in the UNet's self-attention mechanism. This fusion enables the model to focus on regions in the query image that are similar to the support image's annotated object.

For a given layer $l$ in the UNet, let:
$X_s^l$ : Feature map of the support image at layer $l$
$X_q^l$ : Feature map of the query image at layer $l$
The self-attention layer computes:

$$X_q^{l+1} = \text{Attn} \cdot (Q_q, K_{qs}, V_{qs}) \tag{5.69}$$

Where $Q_q = X_q^l W_q$ is the query projection for the query image, $K_{qs} = [K_q, K_s] = \left[X_q^l W_k, X_s^l W_k\right]$ are the concatenated keys from query and support images and $V_{qs} = [V_q, V_s] = \left[X_q^l W_v, X_s^l W_v\right]$ are concatenated values from query and support images.

Thus, the attention mechanism is formulated as:

$$X_q^{l+1} = \text{softmax}\left(\frac{Q_q K_{qs}^T}{\sqrt{d}}\right) V_{qs} \tag{5.70}$$

Let $I_s$ = Support image, $I_q$ = Query image, $M_s$ = The support mask, $M_q$ = The query mask. This is encoded in VQ-VAE to $z_s, z_q, z_{m_s}$ and $z_{m_q}$ respectively. To incorporate support mask information, the support mask $M_s$ is encoded into the latent space $z_s$ and combined with the support image's latent features. The paper evaluates several methods of integrating this mask information:

- Concatenation: $z_s$ and $z_{m_s}$ are concatenated in the channel dimension.
- Multiplication: Mask information is integrated by element-wise multiplication with the support latent.

The chosen approach influences how information about the object of interest in the support image is encoded, aiding the model in distinguishing relevant features for segmentation. During training, DiffewS optimizes for the accurate prediction of the

segmentation mask for the query image. The model uses a modified objective function to train the UNet to produce a mask that closely matches the query mask $M_q$ :

$$L_{fss} = \mathbb{E}_{z_s, z_q, z_{m_s}, z_{m_q}} \left\| z_{m_q} - v_\theta \left( z_s, z_q, z_{m_s} \right) \right\|_2^2 \tag{5.71}$$

Where $v_\theta$ is the modified UNet model that receives the concatenated features from the support and query images (and masks) and outputs the segmentation mask latent for the query image.

**Tokenized Interaction Cross-Attention:** The Tokenized Interaction Cross-Attention approach in DiffewS leverages cross-attention to introduce information from the support image to the query image. Let the query image be the one which we want to segment. The support image, which has an annotated mask of the object of interest, provides the key-value features

To use the support image in cross-attention, it is first encoded into a set of tokens. The support image $I_s$ is passed through a CLIP image encoder, denoted as CLIP $_{\text{img}}$ . The CLIP encoder converts the image into a sequence of tokens (feature vectors):

$$\text{Token}_s = CLIP_{img} \left( I_s \right) \tag{5.72}$$

Where Token $_s \in \mathbb{R}^{Lxd}$ with $L$ being the number of tokens (spatial features) and d the dimension of each token. After encoding the support image, the resulting token sequence Token $_s$ is flattened to match the expected input for the cross-attention layer. Flattening here means that
the 2 D spatial structure of the tokens is reshaped into a single sequence, making it suitable for direct use in cross-attention.

This step is represented as:

$$\text{Flatten } (\text{ Tokens }_s) = [t_1, t_2, \ldots t_L] \in \mathbb{R}^{Lxd}$$

Where $t_i$ are the individual tokens of the support image. The cross-attention mechanism is used to link the query image with the tokenized support image. Here's how the cross-attention operation is set up:

The query features $X_q$ from the query image are projected to create queries Q :

$$Q = X_q W_Q$$

The flattened support image tokens serve as the key (K) and value (V) pairs:

$$K = \text{ Flatten } (\text{ Tokens }_s) W_k, V = \text{ Flatten } (\text{ Tokens }_s) W_V$$

Where $W_k$ and $W_v$ are learned weight matrices for the keys and values, respectively. The crossattention output $X_q^*$ for the query features, influenced by the support image tokens, is computed as:

$$X_q^* = \text{CrossAttn}\left(X_q, \text{Flatten}\left(\text{ Token }_s\right)\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V$$

# 6 Appendices

## 6.1 Code Resources

## 6.2 Suplementary Material

Supplemental proofs and derivations supporting the discussed methodologies.

### 6.2.1 Mutual Information and InfoGANs

Suppose you are trying to guess the weather ($Y$) based on whether someone carries an umbrella ($X$). If someone carrying an umbrella perfectly predicts rain, $X$ and $Y$ are highly dependent, and the mutual information between $X$ and $Y$ will be high. If carrying an umbrella has no relationship with the weather, the mutual information is zero. Formally, Mutual Information measures the amount of information that one random variable $X$ contains about another random variable $Y$. Let X, Y be jointly distributed such that P(x, y) represents their joint probability. Then, mutual information is defined as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{6.1}$$

Where $P(x)$ and $P(y)$ are marginals. If $X$ and $Y$ are independent, $P(x,y) = P(x)P(y)$, so (6.1) is zero. In terms of entropy, one can express the above as:

$$I(X;Y) = H(X) - H(X|Y) \tag{6.2}$$

Where $H(X)$ is the Entropy of $X$ and $H(X|Y)$ is the conditional entropy representing the uncertainty in $X$ after knowing $Y$.

Given the conditional vector $c$ and the generated image $x = G_{gen}$, InfoGANs maximize $I(c;x)$ which measures how much information the generated data $x$ provides

about $c$:

$$\max I(c;x) = H(c) - H(c|x)$$
$$\max I(c;x) = -\int p(c)\log p(c)\,dc + \int \underbrace{p(x)}_{\text{Marginal}} \int p(c|x)\log p(c|x)\,dc\,dx \tag{6.3}$$

Due to the marginal $p(x)$ being computed over space, the direct computation of $H(c|x)$ is not possible. Just like for VAEs, we introduce a introduce a variational lower bound $Q(c,x)$. In terms of this lower bound, the conditional entropy $H(c|x)$ satisfies the inequality:

$$H(c|x) \leq -\int P(c,x)\log Q(c|x)\,dc\,dx, \tag{6.4}$$

Substituting this into (6.3), we get:

$$I(c;x) \geq H(c) + \int P(c,x)\log Q(c|x)\,dc\,dx. \tag{6.5}$$

Assuming $P(c,x) = P(c)P(x|c)$, this can be rewritten as:

$$I(c;x) \geq H(c) + \mathbb{E}_{x\sim G(z,c)}\big[\mathbb{E}_{c'\sim P(c)}\log Q(c'|x)\big].$$
$$I(c;x) \geq \mathbb{E}_{c\sim P(c),x\sim G(z,c)}[\log Q(c|x)] + H(c), \tag{6.6}$$

where $H(c)$ is the entropy of the latent code $c$, and $Q(c|x)$ is the variational approximation of the posterior $P(c|x)$ parametrized by a neural network just as for VAEs. This is the tractable form of the left-side of Info-GAN loss as discussed in the equation (2.19) of the main section.

## 6.3 Additional Figures

Figures and diagrams for extended illustration of key concepts.

# References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In Proceedings of the IEEE/CVF international conference on computer vision, pages 4432–4441, 2019.

[2] Yuval Alaluf, Daniel Garibi, Or Patashnik, Hadar Averbuch-Elor, and Daniel Cohen-Or. Cross-image attention for zero-shot appearance transfer. In ACM SIGGRAPH 2024 Conference Papers, pages 1–12, 2024.

[3] Omri Avrahami, Kfir Aberman, Ohad Fried, Daniel Cohen-Or, and Dani Lischinski. Break-a-scene: Extracting multiple concepts from a single image. In SIGGRAPH Asia 2023 Conference Papers, pages 1–12, 2023.

[4] Omri Avrahami, Thomas Hayes, Oran Gafni, Sonal Gupta, Yaniv Taigman, Devi Parikh, Dani Lischinski, Ohad Fried, and Xi Yin. Spatext: Spatio-textual representation for controllable image generation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 18370–18380, 2023.

[5] Dmitry Baranchuk, Ivan Rubachev, Andrey Voynov, Valentin Khrulkov, and Artem Babenko. Label-efficient semantic segmentation with diffusion models. arXiv preprint arXiv:2112.03126, 2021.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013.

[7] Mingdeng Cao, Xintao Wang, Zhongang Qi, Ying Shan, Xiaohu Qie, and Yinqiang Zheng. Masactrl: Tuning-free mutual self-attention control for consistent image synthesis and editing. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 22560–22570, 2023.

[8] Arantxa Casanova, Marlene Careil, Jakob Verbeek, Michal Drozdzal, and Adriana Romero Soriano. Instance-conditioned gan. Advances in Neural Information Processing Systems, 34:27517–27529, 2021.

[9] Hila Chefer, Yuval Alaluf, Yael Vinker, Lior Wolf, and Daniel Cohen-Or. Attend-and-excite: Attention-based semantic guidance for text-to-image diffusion models. ACM Transactions on Graphics (TOG), 42(4):1–10, 2023.

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. Advances in neural information processing systems, 29, 2016.

[11] Jang Hyun Cho, Utkarsh Mall, Kavita Bala, and Bharath Hariharan. Picie: Unsupervised semantic segmentation using invariance and equivariance in clustering. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 16794–16804, 2021.

[12] Guillaume Couairon, Jakob Verbeek, Holger Schwenk, and Matthieu Cord. Diffedit: Diffusion-based semantic image editing with mask guidance. arXiv preprint arXiv:2210.11427, 2022.

[13] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. Advances in neural information processing systems, 34:8780–8794, 2021.

[14] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. Training-free structured diffusion guidance for compositional text-to-image synthesis. arXiv preprint arXiv:2212.05032, 2022.

[15] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. arXiv preprint arXiv:2208.01618, 2022.

[16] Rinon Gal, Moab Arar, Yuval Atzmon, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. Encoder-based domain tuning for fast personalization of text-to-image models. ACM Transactions on Graphics (TOG), 42(4):1–13, 2023.

[17] Sooyeon Go, Kyungmook Choi, Minjung Shin, and Youngjung Uh. Eye-for-an-eye: Appearance transfer with semantic correspondence in diffusion models. arXiv preprint arXiv:2406.07008, 2024.

[18] Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T Freeman. Unsupervised semantic segmentation by distilling feature correspondences. arXiv preprint arXiv:2203.08414, 2022.

[19] Eric Hedlin, Gopal Sharma, Shweta Mahajan, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. Unsupervised semantic correspondence using stable diffusion. Advances in Neural Information Processing Systems, 36, 2024.

[20] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. arXiv preprint arXiv:2208.01626, 2022.

[21] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. arXiv preprint arXiv:2207.12598, 2022.

[22] Susung Hong, Gyuseong Lee, Wooseok Jang, and Seungryong Kim. Improving sample quality of diffusion models using self-attention guidance. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 7462–7471, 2023.

[23] Ziqi Huang, Tianxing Wu, Yuming Jiang, Kelvin CK Chan, and Ziwei Liu. Reversion: Diffusion-based relation inversion from images. In SIGGRAPH Asia 2024 Conference Papers, pages 1–11, 2024.

[24] Tero Karras. A style-based generator architecture for generative adversarial networks. arXiv preprint arXiv:1812.04948, 2019.

[25] Yunji Kim, Jiyoung Lee, Jin-Hwa Kim, Jung-Woo Ha, and Jun-Yan Zhu. Dense text-to-image generation with attention modulation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 7701–7711, 2023.

[26] Yijun Li, Richard Zhang, Jingwan Lu, and Eli Shechtman. Few-shot image generation with elastic weight consolidation. arXiv preprint arXiv:2012.02780, 2020.

[27] Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. arXiv preprint arXiv:2305.13655, 2023.

[28] Jishnu Mukhoti, Tsung-Yu Lin, Omid Poursaeed, Rui Wang, Ashish Shah, Philip HS Torr, and Ser-Nam Lim. Open vocabulary semantic segmentation with patch aligned contrastive learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 19413–19423, 2023.

[29] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In Proceedings of the IEEE/CVF international conference on computer vision, pages 2085–2094, 2021.

[30] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. ACM Transactions on graphics (TOG), 42(1):1–13, 2022.

[31] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models.

In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10684–10695, 2022.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pages 234–241. Springer, 2015.

[33] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 22500–22510, 2023.

[34] Junjiao Tian, Lavisha Aggarwal, Andrea Colaco, Zsolt Kira, and Mar Gonzalez-Franco. Diffuse attend and segment: Unsupervised zero-shot segmentation using stable diffusion. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3554–3563, 2024.

[35] Omer Tov, Yuval Alaluf, Yotam Nitzan, Or Patashnik, and Daniel Cohen-Or. Designing an encoder for stylegan image manipulation. ACM Transactions on Graphics (TOG), 40(4):1–14, 2021.

[36] Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 1921–1930, 2023.

[37] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103, 2008.

[38] Andrey Voynov, Qinghao Chu, Daniel Cohen-Or, and Kfir Aberman. p+: Extended textual conditioning in text-to-image generation. arXiv preprint arXiv:2303.09522, 2023.

[39] Yanchao Yang and Stefano Soatto. Fda: Fourier domain adaptation for semantic segmentation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 4085–4095, 2020.

[40] Hu Yu, Hao Luo, Fan Wang, and Feng Zhao. Uncovering the text embedding in text-to-image diffusion models. arXiv preprint arXiv:2404.01154, 2024.

[41] Muzhi Zhu, Yang Liu, Zekai Luo, Chenchen Jing, Hao Chen, Guangkai Xu, Xinlong Wang, and Chunhua Shen. Unleashing the potential of the diffusion model in few-shot semantic segmentation. arXiv preprint arXiv:2410.02369, 2024.